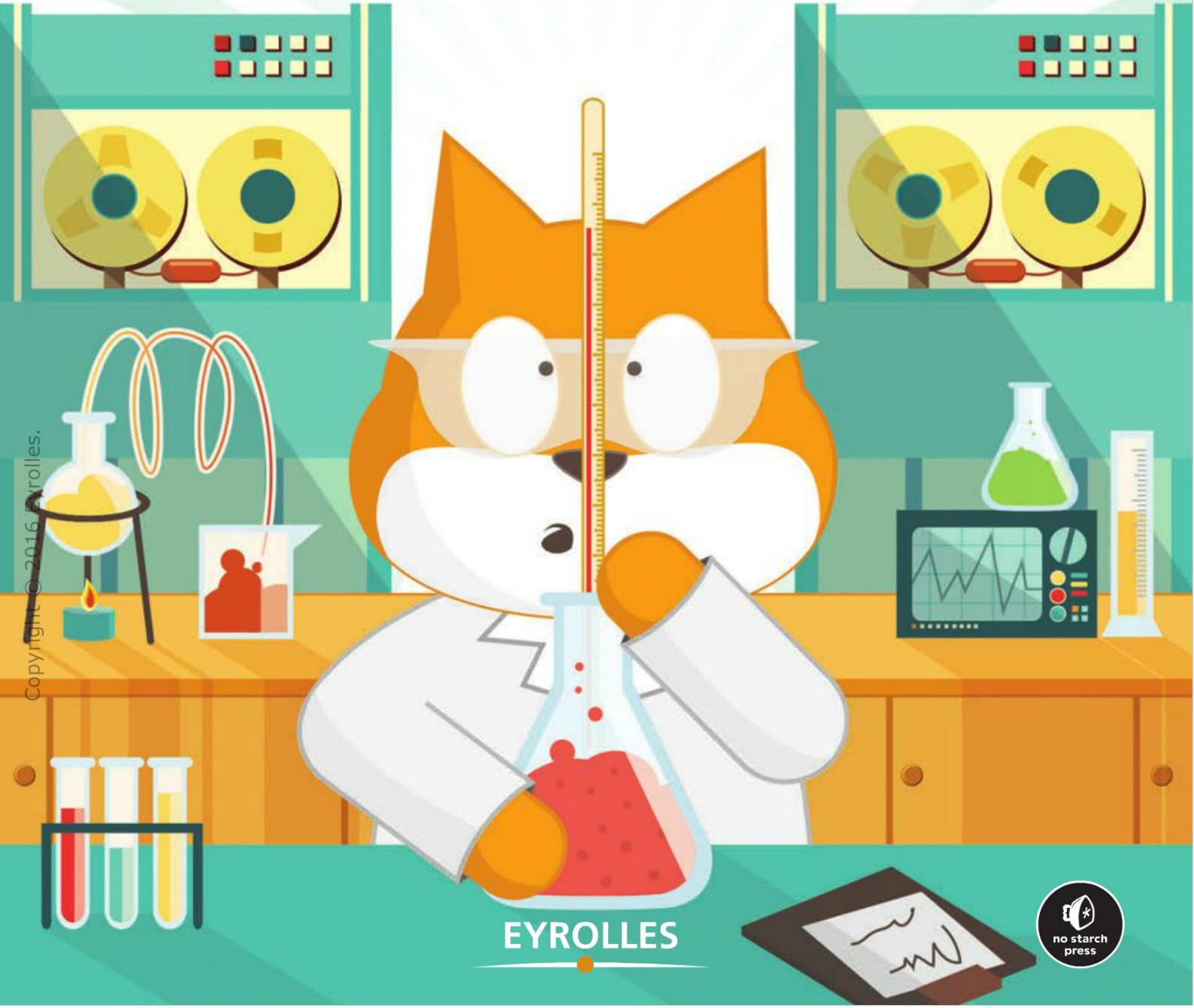
Majed Marji

Le grand livre de SCRATCH

Une initiation visuelle à la programmation mêlant jeux, art et science



Copyright © 2016 Eyrolles.

EYROLLES

no starch press

Le grand livre de SCRATCH

Apprendre à programmer en s'amusant !

Scratch est un environnement de programmation simple, ludique et gratuit où il s'agit de relier des blocs de code pour construire des programmes. Il est très utilisé pour apprendre aux enfants les bases de la programmation, mais il peut convenir à tout grand débutant en informatique.

Pratique et didactique, cet ouvrage de référence couvre tous les aspects de Scratch, de la prise en main du langage jusqu'à la maîtrise de ses instructions les plus pointues. Il explique les grands principes de la programmation (variables, boucles, listes, opérateurs logiques, entrées-sorties, etc.), en les illustrant par de nombreux exemples et exercices puisés dans des domaines aussi variés que les mathématiques, la biologie, les arts appliqués, ou encore les jeux.

À qui s'adresse ce livre ?

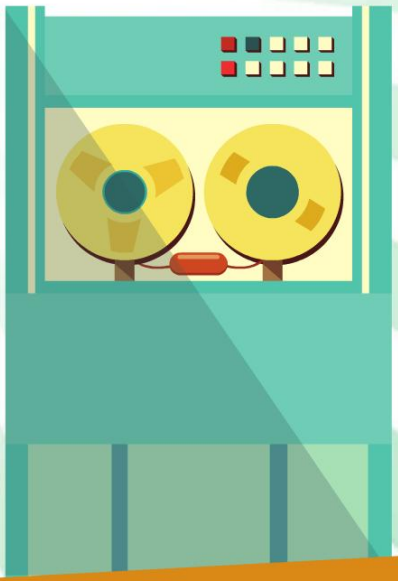
- Aux enfants (dès 10 ans) et leurs parents
- Aux enseignants, formateurs, éducateurs et animateurs d'ateliers Scratch
- Aux grands débutants en programmation

Sur www.éditions-eyrolles.com/go/grandlivrescratch

Téléchargez le code source des exemples et les solutions des exercices.

À propos de l'auteur

Docteur en génie électrique de l'université de Wayne State (Michigan), Majed Marji est également titulaire d'un MBA en stratégie d'entreprise de l'université de Davenport (Michigan). Il possède plus de 15 années d'expérience dans le secteur de la construction automobile, au sein duquel il a développé un grand nombre d'applications informatiques. Il est en outre professeur auxiliaire dans le département Génie électrique de l'université de Wayne State.





Copyright © 2016 Eyrolles.

Dans la même collection







www.editions-eyrolles.com

no starch press

Le grand livre de SCRATCH

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Majed Marji

Le grand livre de SCRATCH

Copyright © 2016 Eyrolles.

no starch press

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Copyright © 2014 by Majed Marji. Title of English-language original: *Learn to Program with Scratch*, ISBN 978-1-59327-543-3, published by No Starch Press.

French-language edition copyright © 2017 by Eyrolles. All rights reserved.

Traduction autorisée de l'ouvrage en langue anglaise intitulé *Learn to Pogram with Scratch* de Majed Marji (ISBN 978-1-59327-543-3), publié par No Starch Press.

Adapté de l'anglais par Samuel Delalez.

Copyright © 2016 Eyrolles.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre français d'exploitation du droit de copie, 20, rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2017, ISBN : 978-2-212-11812-4

REMERCIEMENTS

Bien que la couverture de ce livre ne mentionne qu'un seul auteur, beaucoup de personnes ont participé à sa création. J'aimerais remercier les nombreux professionnels de No Starch Press qui ont contribué à ce travail, particulièrement mon éditrice, Jennifer Griffith-Delgado, et mon éditrice de production, Alison Law, pour leurs importantes contributions. Leur expertise a grandement amélioré cet ouvrage et leur quête de l'excellence se retrouve dans chaque page. Par ailleurs, merci également à Paula L. Fleming et Serena Yang pour leur travail sur ce livre.

Je suis aussi très reconnaissant des retours précieux de Tyler Watts, l'éditeur technique. Ses remarques pertinentes ont amélioré le contenu de ce livre dans bien des cas.

Enfin, je tiens à remercier ma femme, Marina, et mes deux fils, Asad et Karam, qui m'ont soutenu durant ce long projet. Ils ont tout fait pour me laisser le temps et l'espace dont j'avais besoin. Je peux peut-être à présent rattraper tout ce que j'ai manqué !

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

AVANT-PROPOS

Scratch est un langage de programmation visuelle qui offre un riche environnement d'apprentissage, adapté à tous les âges. Il permet de créer des projets multimédias et interactifs, comme des histoires animées, des rapports de lecture, des programmes scientifiques ou encore des jeux.

L'environnement de programmation visuelle de Scratch vous donne la possibilité d'explorer des domaines qui vous seraient autrement inaccessibles. Il fournit un ensemble complet d'outils multimédias que vous utiliserez pour créer de superbes applications bien plus facilement qu'avec n'importe quel autre langage de programmation.

Scratch aide, de bien des manières, à développer des compétences de résolution de problèmes qui vous seront utiles dans la vie de tous les jours, pas uniquement pour la programmation. L'environnement fournit des retours immédiats, grâce auxquels vous vérifiez votre logique rapidement et facilement. La structure visuelle simplifie grandement l'écriture de vos programmes et le perfectionnement de votre logique. Par essence, Scratch rend

Copyright © 2016 Eyrolles.

les principes de l'informatique accessibles à tous. Il rend l'apprentissage foncièrement motivant et il encourage la quête de connaissances par soi-même via l'exploration et la découverte. Vous ne serez limité que par votre créativité et votre imagination.

Beaucoup de livres affirment enseigner la programmation avec Scratch. Pour la plupart, ils ciblent de très jeunes lecteurs et ne présentent que quelques applications simples qui les guideront à travers l'interface utilisateur de Scratch. Ils traitent plus de Scratch que de la programmation en elle-même. Ce livre, en revanche, a pour but d'enseigner les concepts fondamentaux de la programmation en utilisant Scratch comme un outil, mais aussi de dévoiler les puissantes capacités d'enseignement et d'apprentissage de ce langage.

À qui s'adresse ce livre ?

Ce livre s'adresse à tous ceux qui souhaitent étudier l'informatique. Il enseigne les principes fondamentaux de la programmation et il peut être utilisé comme manuel scolaire pour collégiens et lycéens, ou comme guide pour autodidactes. Il est également susceptible d'être utilisé à l'université pour enseigner les concepts élémentaires de la programmation à des étudiants issus de différentes formations, ou pour servir de manuel d'accompagnement en guise d'introduction à un module de programmation.

Les enseignants souhaitant utiliser Scratch dans une classe de primaire bénéficieront également d'une compréhension plus approfondie de la programmation. Ils développeront les compétences nécessaires pour sensibiliser leurs élèves à Scratch de la manière qui correspondra le mieux à leurs besoins.

Aucune expérience en programmation n'est nécessaire à la prise en main de ce livre et les notions de mathématiques présentées ici dépasseront très rarement le niveau du lycée. Sachez aussi que certaines simulations avancées pourront être laissées de côté sans créer de lacunes.

Recommandations au lecteur

La beauté de la programmation est qu'elle rend capable de créer. Il suffit d'une idée et de quelques heures sur un clavier d'ordinateur pour qu'un nouveau logiciel prenne vie ! Comme n'importe quelle nouvelle compétence cependant, la programmation requiert de l'entraînement. Au cours de votre apprentissage, vous commettrez sans doute des erreurs, mais n'abandonnez surtout pas. Prenez du temps pour réfléchir aux concepts et expérimentez différentes techniques jusqu'à ce que vous les maîtrisiez, avant de passer à quelque chose de nouveau.

Concept de l'ouvrage

Ce livre va vous faire découvrir les grands principes de la programmation à travers la résolution de problèmes pratiques. Avec cette approche, j'espère cultiver l'imagination des lecteurs et rendre la programmation informatique accessible à tous.

Copyright © 2016 Eyrolles.

Cet ouvrage est donc clairement axé sur les projets. Je présenterai de manière détaillée les différents concepts de la programmation, puis nous développerons ensemble un certain nombre d'applications pour les illustrer. L'accent sera mis sur la résolution de problèmes plutôt que sur les caractéristiques particulières de Scratch.

Les exemples présentés dans ce livre montrent la grande palette de connaissances que vous pourrez explorer avec Scratch. Ces exemples ont été sélectionnés avec attention pour expliquer les concepts de la programmation et vous montrer comment utiliser Scratch afin de perfectionner votre compréhension d'autres sujets.

Les exercices « Faites le test » et les problèmes figurant à la fin de chaque chapitre sont conçus pour mettre vos compétences de programmeur à l'épreuve.

Structure du livre

Les trois premiers chapitres de ce livre introduisent Scratch, en le présentant comme un puissant outil de dessin de formes géométriques et de création d'applications multimédias. Ils vous aideront à démarrer rapidement et facilement, tandis que le reste de l'ouvrage se focalise sur les outils de programmation que propose Scratch.

Le chapitre 1, **Premiers pas**, présente l'environnement de programmation de Scratch, les blocs de commande disponibles et la procédure permettant de créer des programmes simples.

Le chapitre 2, **Mouvements et dessins**, porte sur les commandes de mouvement et les fonctions de dessin de Scratch.

Le chapitre 3, **Apparences et sons**, traite des commandes graphiques et sonores.

Le chapitre 4, **Procédures**, explique comment écrire des programmes structurés et modulaires grâce aux procédures.

Le chapitre 5, **Variables**, explore la façon dont les variables peuvent être utilisées pour enregistrer des informations. Ce chapitre explique également comment obtenir des réponses des utilisateurs après leur avoir posé des questions, ce qui vous permettra de construire un large panel de programmes interactifs.

Le chapitre 6, **Prendre des décisions**, décrit les différentes manières de prendre des décisions et de contrôler le cheminement d'un programme.

Le chapitre 7, **Répétitions. Une étude approfondie des boucles**, traite en détail des structures de répétitions disponibles dans Scratch et explique comment les utiliser avec des exemples concrets.

Le chapitre 8, **Traitement des chaînes de caractères**, présente un ensemble de programmes utiles pour manipuler le type de données « chaîne de caractères ».

Copyright © 2016 Eyrolles.

Le chapitre 9, **Listes**, porte sur les listes, des contenants servant à stocker des éléments, et montre comment les utiliser pour créer des programmes puissants.

Chaque chapitre contient en outre plusieurs exemples de projets qui peuvent servir de guide à la création d'applications similaires dans de nombreux cadres d'apprentissage. Quand vous aurez fini ce livre, vous serez capable d'aborder seul n'importe quel projet de programmation !

Conventions utilisées

Nous utilisons des styles de texte particuliers pour faire référence aux textes de l'interface de Scratch.

- Les noms des blocs de Scratch sont dans ce style : **quand le drapeau vert est cliqué**.
- **Les noms des lutins et des variables sont dans ce style :** Balle.

Nomfichier.sb2

Les fichiers dont vous aurez besoin en lisant certaines sections sont indiqués dans la marge (voir l'exemple sur le côté) et les exercices « Faites le test » sont présentés de la manière suivante :

> **FAITES LE TEST**
>
> Voici quelque chose que vous pouvez tester.

Ressources en ligne

Rendez-vous à l'adresse *http://www.editions-eyrolles.com/go/grandlivrescratch* pour télécharger les ressources du livre, qui se composent des documents suivants :

- **Scripts**. Ce dossier contient tous les scripts mentionnés dans l'ouvrage.
- **Solutions**. Ce dossier contient les solutions à tous les problèmes et exercices du livre.
- **Ressources annexes**. Ce dossier contient trois fichiers PDF fournissant des informations plus approfondies (en anglais) sur des sujets particuliers (l'éditeur d'images, les fonctions mathématiques et les dessins de formes géométriques).
- **Applications bonus**. Ce dossier contient d'autres applications Scratch que vous pourrez étudier par vous-même. Le fichier ApplicationsBonus.pdf vous en fournit des explications détaillées (en anglais).

Copyright © 2016 Eyrolles.

TABLE DES MATIÈRES

1
PREMIERS PAS **1**

Qu'est-ce que Scratch ? . . . 2
Faites le test 1-1 . . . 3
L'environnement de programmation de Scratch . . . 3
La scène . . . 4
Faites le test 1-2 . . . 4
La liste des lutins . . . 5
Faites le test 1-3 . . . 5
L'onglet des blocs . . . 6
Faites le test 1-4 . . . 7
La zone de scripts . . . 7
Faites le test 1-5 . . . 8
Faites le test 1-6 . . . 9
L'onglet Costumes . . . 9
Faites le test 1-7 . . . 10
L'onglet Sons . . . 10
Faites le test 1-8 . . . 10
L'onglet Arrière-plans . . . 11
Faites le test 1-9 . . . 11
Les informations du lutin . . . 11
La barre d'outils . . . 12
L'éditeur d'images . . . 13
Définir le centre d'une image . . . 13
Faites le test 1-10 . . . 14
Définir une couleur transparente . . . 14
Votre premier jeu Scratch . . . 15
Étape 1. Préparez l'arrière-plan . . . 16
Étape 2. Ajoutez la raquette et la balle . . . 16
Étape 3. Démarrez le jeu et déplacez vos lutins . . . 17
Faites le test 1-11 . . . 18
Étape 4. Mettez de l'ambiance avec du son . . . 19
Les blocs de Scratch : un aperçu . . . 20
Opérateurs arithmétiques et fonctions . . . 21
Opérateurs arithmétiques . . . 21
Nombres aléatoires . . . 21
Fonctions mathématiques . . . 22
Résumé . . . 22
Problèmes . . . 23

2
MOUVEMENTS ET DESSINS **25**

Utilisation des commandes de mouvement . . . 25
Mouvement absolu . . . 26
Faites le test 2-1 . . . 27

Copyright © 2016 Eyrolles.

Mouvement relatif . . . 27
La direction et les costumes . . . 29
Faites le test 2-2 . . . 30
Autres commandes de mouvement . . . 30
Dessiner avec les commandes de stylo . . . 31
Faites le test 2-3 . . . 31
Faites le test 2-4 . . . 32
Le pouvoir des répétitions . . . 33
Faites le test 2-5 . . . 34
Carrés rotatifs . . . 34
Faites le test 2-6 . . . 35
L'estampillage . . . 35
Faites le test 2-7 . . . 35
Projets Scratch . . . 36
Prenez l'argent . . . 36
Le chronomètre de Scratch . . . 39
Attraper des pommes . . . 39
Plus de détails sur les lutins clonés . . . 42
Résumé . . . 43
Problèmes . . . 44

3
APPARENCE ET SONS **47**

La palette Apparence . . . 48
Animer en changeant de costume . . . 48
Faites le test 3-1 . . . 49
Des lutins qui pensent et qui parlent . . . 50
Faites le test 3-2 . . . 50
Effets graphiques . . . 50
Taille et visibilité . . . 51
Faites le test 3-3 . . . 52
Plans . . . 52
Faites le test 3-4 . . . 53
La palette Sons . . . 53
Jouer un fichier audio . . . 53
Jouer des percussions et d'autres sons . . . 54
Composer de la musique . . . 55
Contrôle du volume sonore . . . 55
Faites le test 3-5 . . . 56
Régler le tempo . . . 56
Faites le test 3-6 . . . 56
Projets Scratch . . . 56
Danser sur scène . . . 57
Feu d'artifice . . . 60
Résumé . . . 62
Problèmes . . . 63

Copyright © 2016 Eyrolles.

4
LES PROCÉDURES 67

Émission et réception de messages 68
Envoyer et recevoir des messages 69
L'envoi de messages pour la coordination de plusieurs lutins 70
Créer de longs programmes en petites étapes 72
Créer des procédures avec l'émission de messages 73
Construire vos propres blocs 75
Transmettre des paramètres aux blocs personnalisés 77
Exécuter sans rafraîchissement de l'écran 77
Paramètre ou argument ? 80
Faites le test 4-1 81
Utiliser des procédures imbriquées 81
Faites le test 4-2 84
Travailler avec les procédures 84
Processus descendant : diviser des programmes en procédures 84
Faites le test 4-3 86
Processus montant : construire un programme à partir de procédures 87
Résumé 89
Problèmes 89

5
LES VARIABLES 91

Les types de données dans Scratch 92
Quelle forme choisir ? 92
Conversion automatique de types de données 93
Introduction aux variables 94
Qu'est-ce qu'une variable ? 94
Créer et utiliser des variables. 97
Nommer les variables 98
Faites le test 5-1 100
La portée des variables 100
Le type de donnée d'une variable 101
Modifier les variables. 102
Faites le test 5-2 104
Les variables dans les clones 105
Les moniteurs de variables 107
Utiliser les moniteurs de variables dans des applications 108
Simulation de la loi d'Ohm 109
Faites le test 5-3 110
Démonstration d'un circuit en série 110
Faites le test 5-4 112
Visualiser le volume et l'aire d'une sphère. 112
Faites le test 5-5 114
Dessiner une rosace à n pétales 114
Faites le test 5-6 116
Modéliser la répartition des graines de tournesol 116
Faites le test 5-7 118
Obtenir des données entrées par les utilisateurs 118
Lire un nombre 118

Copyright © 2016 Eyrolles.

Lire des caractères . . . 119
Effectuer des opérations arithmétiques . . . 119
Résumé . . . 120
Problèmes . . . 121

6
PRENDRE DES DÉCISIONS 123

Les opérateurs de comparaison . . . 124
Les booléens dans le monde réel . . . 125
Évaluation des expressions booléennes . . . 126
Comparer des lettres et des chaînes de caractères . . . 126
Les structures de décision . . . 128
Le bloc si . . . 128
Utiliser les variables comme drapeaux . . . 129
Le bloc si/sinon . . . 130
Blocs si et si/sinon imbriqués . . . 132
Programmes pilotés par des menus . . . 132
Les opérateurs logiques . . . 134
L'opérateur et . . . 135
L'opérateur ou . . . 135
L'opérateur non . . . 136
Utiliser les opérateurs logiques pour tester des intervalles . . . 137
Comparer des nombres décimaux . . . 139
Projets Scratch . . . 140
Devine mes coordonnées . . . 140
Faites le test 6-1 . . . 142
Jeu de classification de triangles . . . 142
Faites le test 6-2 . . . 145
Suiveur de ligne . . . 145
Faites le test 6-3 . . . 147
Équation d'une droite . . . 147
Faites le test 6-4 . . . 151
Autres applications . . . 151
Résumé . . . 151
Problèmes . . . 152

7
RÉPÉTITIONS : UNE ÉTUDE APPROFONDIE DES BOUCLES 155

Plus de blocs de boucles . . . 156
Le bloc répéter jusqu'à . . . 157
Faites le test 7-1 . . . 158
Construire un bloc si infini . . . 158
Faites le test 7-2 . . . 159
Les commandes d'arrêt (stop) . . . 160
Faites le test 7-3 . . . 161
Mettre fin à une boucle . . . 162
Valider les entrées utilisateur . . . 162
Les compteurs . . . 164
Vérifier un mot de passe . . . 164
Faites le test 7-4 . . . 165
Compter avec un incrément constant . . . 165

Copyright © 2016 Eyrolles.

Retour sur les boucles imbriquées . . . 167
Faites le test 7-5 . . . 169
Récursivité : des procédures qui s'appellent elles-mêmes . . . 169
Faites le test 7-6 . . . 171
Projets Scratch . . . 171
Horloge analogique . . . 171
Faites le test 7-7 . . . 173
Le jeu du Chasseur d'oiseaux . . . 173
Faites le test 7-8 . . . 176
Simulation de chute libre . . . 177
Faites le test 7-9 . . . 178
Simulation du mouvement d'un projectile . . . 179
Faites le test 7-10 . . . 182
Autres applications . . . 182
Résumé . . . 182
Problèmes . . . 183

8
TRAITEMENT DES CHAÎNES DE CARACTÈRES 185

Retour sur les chaînes de caractères . . . 186
Compter les caractères d'un certain type dans une chaîne . . . 186
Comparer des chaînes de caractères . . . 188
Faites le test 8-1 . . . 189
Exemples de manipulation de chaînes de caractères . . . 189
Angagelille ecretsille . . . 190
Faites le test 8-2 . . . 191
Corrige mon orthographe . . . 191
Faites le test 8-3 . . . 193
Réordonne . . . 193
Projets Scratch . . . 196
Tir . . . 196
Faites le test 8-4 . . . 198
Convertisseur binaire-décimal . . . 199
Faites le test 8-5 . . . 200
Faites le test 8-6 . . . 202
Le jeu du Pendu . . . 202
Faites le test 8-7 . . . 207
Professeur de fractions . . . 207
Faites le test 8-8 . . . 212
Résumé . . . 212
Problèmes . . . 213

9
LES LISTES 215

Les listes dans Scratch . . . 216
Créer des listes . . . 216
Faites le test 9-1 . . . 218
Les commandes de listes . . . 218
Faites le test 9-2 . . . 220
Vérification des limites . . . 221
Les listes dynamiques . . . 222
Remplir des listes avec des entrées utilisateur . . . 222

Copyright © 2016 Eyrolles.

Création d'un histogramme 223
Faites le test 9-3 226
Listes numériques 226
Trouver le maximum et le minimum 226
Faites le test 9-4 227
Trouver la moyenne 227
Localiser et trier les éléments d'une liste. 228
Recherche linéaire 228
Fréquence d'occurrence. 229
Tri à bulle 231
Faites le test 9-6 231
Trouver la médiane. 233
Faites le test 9-7 234
Projets Scratch 235
Le poète 235
Jeu de classification de quadrilatères 236
Faites le test 9-8 236
Faites le test 9-9 238
Magicien des mathématiques 238
Quizz sur l'anatomie d'une fleur 242
Faites le test 9-10 242
Autres applications. 243
Résumé 244
Faites le test 9-11 244
Problèmes 244

ANNEXE
PARTAGE ET COLLABORATION 247

Créer un compte Scratch 247
Utilisation du sac à dos 250
Partage de vos propres projets 252
Démarrer un nouveau projet. 252
Remixer un projet 253
La page de projet. 253
Partager votre projet 254

INDEX 255

À PROPOS DE L'AUTEUR ET DU VALIDATEUR TECHNIQUE 261

Copyright © 2016 Eyrolles.

1

PREMIERS PAS

Avez-vous déjà éprouvé le souhait de créer vos propres jeux vidéo, histoires animées, tutoriels ou simulations scientifiques ? Scratch est un langage de programmation graphique qui permet facilement de créer ce type d'applications. Dans ce chapitre introductif, vous allez :

- explorer l'environnement de programmation de Scratch ;
- vous familiariser avec les différents types de blocs de commande ;
- créer votre premier jeu dans Scratch.

Lorsque vous créez une application Scratch, vous pouvez la sauvegarder sur votre ordinateur ou la télécharger sur le site Internet de Scratch, où d'autres utilisateurs pourront la commenter et la réutiliser dans de nouveaux projets.

Vous êtes prêt ? Alors c'est parti !

Copyright © 2016 Eyrolles.

Qu'est-ce que Scratch ?

Un programme informatique consiste en une suite d'instructions qui indiquent à l'ordinateur ce qu'il doit faire. Ces instructions sont écrites grâce à un *langage de programmation* ; c'est là que Scratch entre en jeu.

Pour la plupart, les langages de programmation sont *à base de texte*, ce qui signifie que vous devez donner à l'ordinateur des instructions qui ressemblent à une forme d'anglais crypté. Par exemple, pour afficher « Bonjour ! » à l'écran, vous écririez :

```
Print('Bonjour !')                              (en langage Python)
Std ::cout << "Bonjour !" << std :endl ;         (en langage C++)
System.out.print("Bonjour !") ;                  (en langage Java)
```

L'apprentissage de ces langages et la compréhension de leurs règles syntaxiques peuvent être difficile pour les débutants. Scratch, quant à lui, est un langage de programmation *visuelle*. Il a été développé dans le laboratoire multimédia de l'Institut de technologie du Massachusetts (MIT) afin de simplifier la programmation et de rendre son apprentissage plus amusant.

Dans Scratch, vous ne taperez aucune commande compliquée : vous connecterez des blocs graphiques entre eux pour créer vos programmes. Si ce n'est pas clair pour vous, regardez le programme simple présenté sur la figure 1-1.

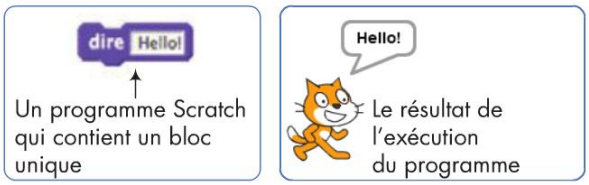


Figure 1-1 : Lorsque vous exécutez ce bloc de Scratch, le chat dit « Hello! » dans une bulle de bande dessinée.

Le chat que vous voyez sur la figure 1-1 est ce que l'on appelle un *lutin*. Les lutins comprennent et exécutent les suites d'instructions que vous leur donnez. Le bloc violet sur la gauche indique au chat d'afficher « Hello! » dans une bulle de bande dessinée. Une grande partie des applications que vous créerez dans ce livre contiendront plusieurs lutins et vous utiliserez des blocs pour les faire bouger, tourner, dire des choses, jouer de la musique, faire des maths, etc.

Vous programmez dans Scratch en assemblant ces blocs de couleur, comme vous le feriez avec des briques de Lego. Les piles de blocs que vous créez s'appellent des *scripts*. Par exemple, la figure 1-2 montre un script qui change quatre fois la couleur d'un lutin.

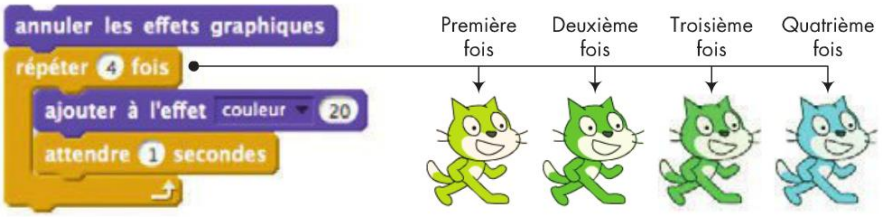


Figure 1-2 : Utilisation d'un script qui change la couleur du lutin Chat

Copyright © 2016 Eyrolles.

Ce script attend une seconde entre chaque changement de couleur, et les quatre chats que vous voyez ici montrent la nouvelle couleur du lutin après chaque changement.

FAITES LE TEST 1-1

Bien que nous n'ayons pas encore parlé des blocs de la figure 1-2, lisez-les, observez leurs formes et essayez de deviner par quelles étapes le script est passé pour faire changer le chat de couleur (indice : le premier bloc violet renvoie le chat à sa couleur originale). À votre avis, que se passerait-il si nous retirions le bloc **attendre** du script ?

Ce livre parcourt Scratch 2, qui est sorti en mai 2013. Cette version vous permet de créer des projets directement dans votre navigateur Internet et vous évite d'installer un programme sur votre ordinateur. Le contenu de ce livre s'appuiera sur l'interface Internet de Scratch.

Maintenant que vous en savez un peu plus sur ce langage, il est temps de commencer notre étude de la programmation et d'apprendre à nous servir de Scratch.

L'environnement de programmation de Scratch

Pour démarrer Scratch, rendez-vous sur le site Internet *http://scratch.mit.edu/* et cliquez en haut à gauche sur le lien *ESSAIE-LE*. Vous vous retrouverez alors sur l'interface d'édition de projet de Scratch (figure 1-3).

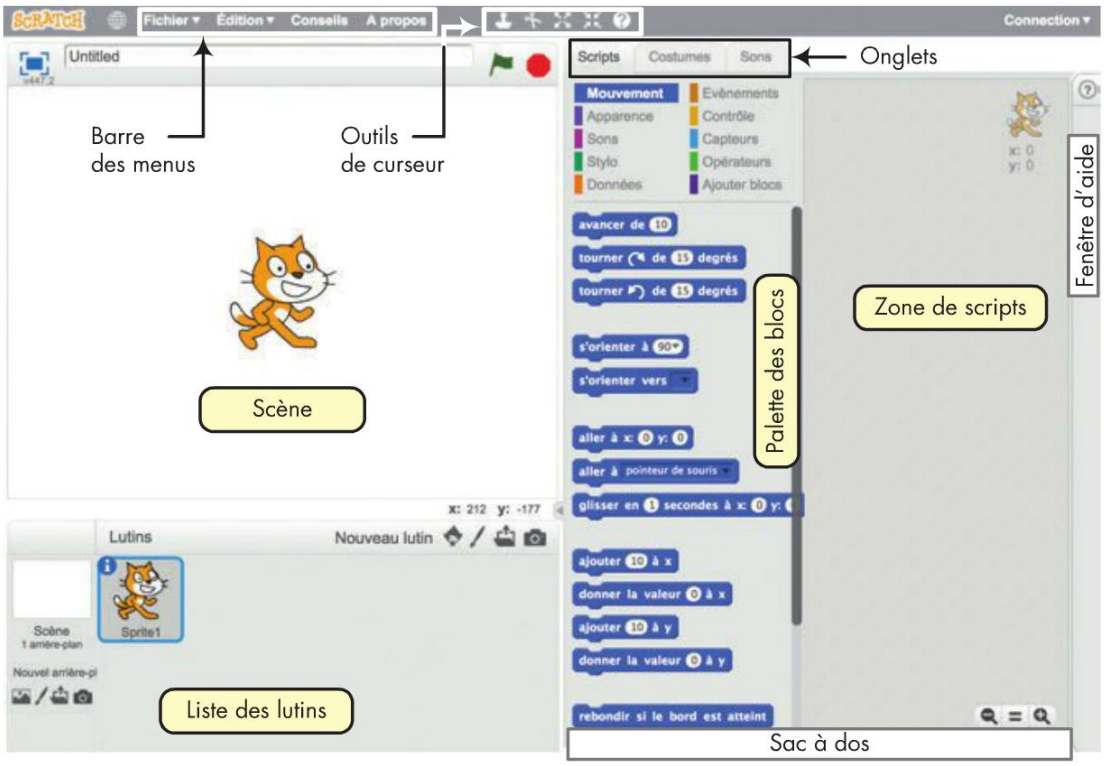


Figure 1-3 : L'interface utilisateur de Scratch, où vous construirez vos programmes

Copyright © 2016 Eyrolles.

Dans une fenêtre unique, vous devriez au moins voir les trois volets suivants : la scène (en haut à gauche), la liste des lutins (en bas à gauche) et l'onglet **Scripts** (à droite), qui contient l'onglet des blocs et la zone de script. Le volet de droite contient deux onglets supplémentaires, **Costumes** et **Sons**, qui seront abordés plus tard dans cette partie. Si vous êtes connectés à un compte sur le site de Scratch, vous devriez également voir le **Sac à dos** (en bas à droite), qui contient des boutons pour partager votre projet et utiliser des lutins et des scripts de projets existants.

Regardons rapidement ce qui se trouve dans ces trois volets.

La scène

La scène est l'endroit où vos lutins se déplacent, dessinent et interagissent. Elle a une largeur de 480 pas et une hauteur de 360 pas (figure 1-4). Le centre de la scène a une coordonnée x de 0 et une coordonnée y de 0.

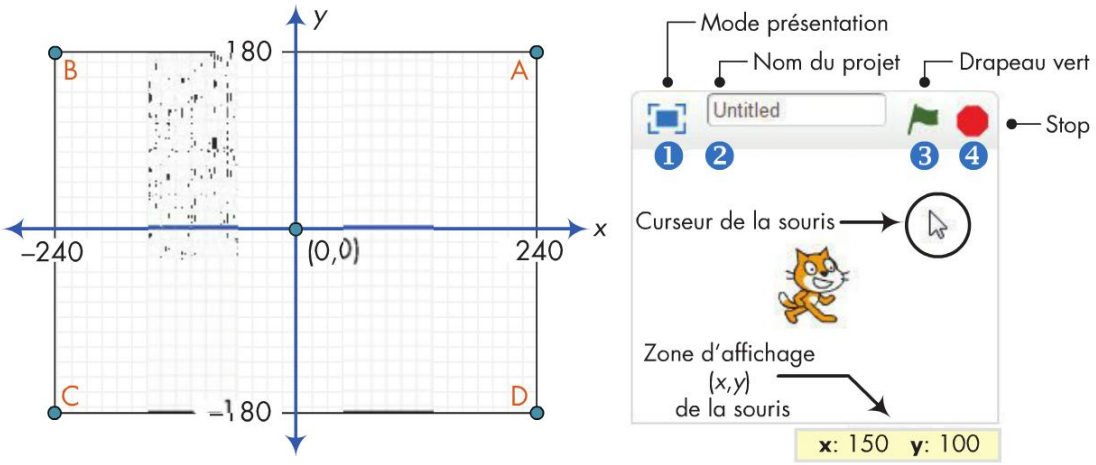


Figure 1-4 : La scène est semblable à une grille de coordonnées avec le point (0,0) au centre.

Vous trouvez les coordonnées (x,y) de n'importe quel point de la scène en y déplaçant le curseur de la souris et en regardant les valeurs dans la zone d'affichage (x,y) de la souris, située directement sous la scène.

La petite barre située au-dessus de la scène contient plusieurs commandes. L'icône du mode Présentation ❶ cache tous les scripts et outils de programmation et agrandit la scène à la quasi-totalité de votre écran. La zone de texte ❷ affiche le nom du projet actuel. Les icônes du drapeau vert ❸ et du panneau stop ❹ servent respectivement à démarrer et arrêter votre programme.

FAITES LE TEST 1-2

Déplacez la souris dans la scène et observez la zone d'affichage de la souris. Que se passe-t-il lorsque vous déplacez la souris hors de la scène ? À présent, passez en mode Présentation et observez la façon dont change l'écran. Cliquez sur l'icône en haut à gauche de l'écran ou appuyez sur la touche ESC du clavier pour quitter le mode Présentation.

Copyright © 2016 Eyrolles.

La liste des lutins

La liste des lutins affiche les noms et les images miniatures de tous les lutins de votre projet. Les nouveaux projets démarrent avec une scène blanche et un lutin portant un costume de chat (figure 1-5).

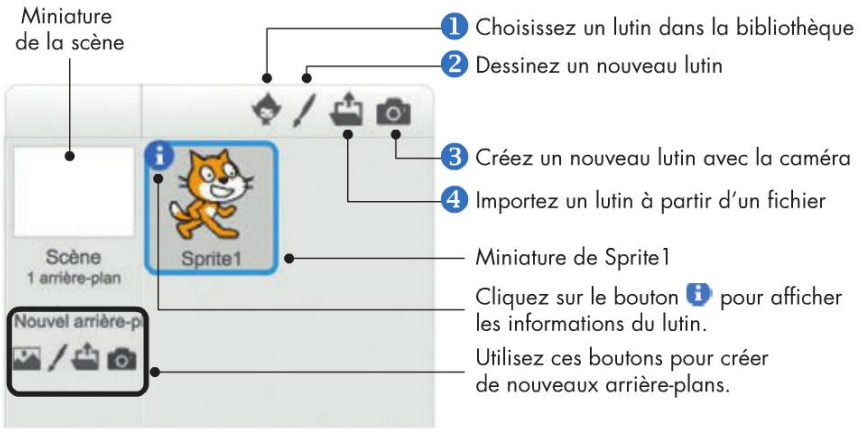


Figure 1-5 : La liste des lutins d'un nouveau projet

Les boutons situés au dessus de la liste des lutins vous permettent d'ajouter de nouveaux lutins à votre projet à partir de l'un des quatre emplacements suivants : la bibliothèque de lutins de Scratch ❶, l'éditeur d'images intégré ❷ (où vous pouvez dessiner votre propre costume), une caméra intégrée à votre ordinateur ❸ ou un fichier sur votre ordinateur ❹.

FAITES LE TEST 1-3

Ajoutez de nouveaux lutins à votre projet en utilisant quelques-uns des boutons. Réorganisez les lutins dans la liste en faisant glisser les images miniatures correspondantes.

Chaque lutin de votre projet possède ses propres scripts, costumes et sons. Sélectionnez n'importe lequel pour voir ce qu'il contient. Vous pouvez soit (1) cliquer sur la miniature du lutin dans la liste, soit (2) double-cliquer sur le lutin lui-même dans la scène. La miniature de celui qui est sélectionné est toujours en surbrillance et entourée en bleu. Vous pouvez alors accéder à ses scripts, costumes et sons en cliquant sur l'un des trois onglets situés dans la zone de scripts. Nous observerons le contenu de ces onglets plus tard. Pour l'instant, effectuez un clic droit (ou Ctrl + clic si vous utilisez un Mac) sur la miniature du lutin Chat pour voir le menu contextuel présenté sur la figure 1-6.

L'option **dupliquer** ❶ copie le lutin en donnant un nom différent à la copie. Vous pouvez retirer un lutin de votre projet avec l'option **supprimer** ❷ ou l'exporter au format *.sprite2* sur votre ordinateur en utilisant l'option **enregistrer localement comme fichier** ❸. Pour importer et exporter vers un autre projet, cliquez sur le bouton **importer le lutin depuis un fichier** présenté à la figure 1-5. L'option **cacher** ❹ rend le lutin visible ou invisible sur la scène.

Copyright © 2016 Eyrolles.

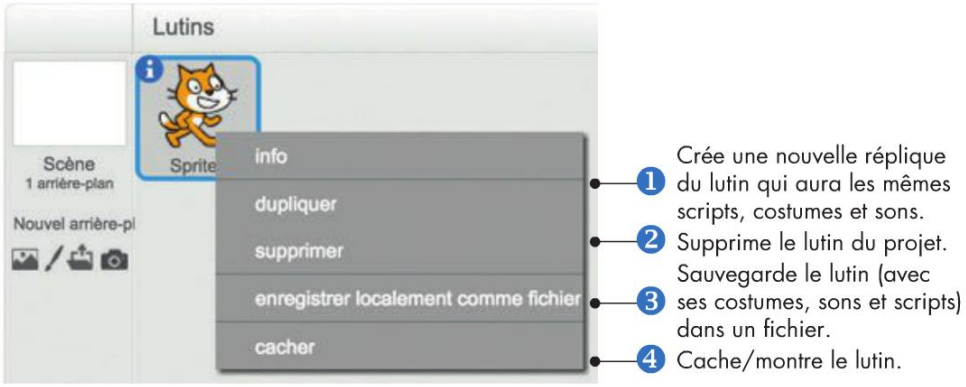


Figure 1-6 : Un clic droit sur la miniature d'un lutin fait apparaître ce menu contextuel très utile.

La liste des lutins montre également une miniature pour la scène sur la gauche (figure 1-6). La scène a ses propres suites d'instructions, images et sons. Le fond d'écran que vous voyez sur la scène est appelé *arrière-plan*. Par défaut, lorsque vous démarrez un nouveau projet, l'arrière-plan est constitué d'un fond blanc, mais vous pouvez ajouter de nouvelles images grâce à l'un des quatre boutons se trouvant sous la miniature de la scène. Cliquez sur la miniature de la scène pour voir et éditer les scripts, arrière-plans et sons qui lui sont associés.

L'onglet des blocs

Les blocs de Scratch sont divisés en 10 catégories (palettes) : *Mouvement*, *Apparence*, *Sons*, *Stylo*, *Données*, *Événements*, *Contrôle*, *Capteurs*, *Opérateurs* et *Ajouter blocs*. Ils obéissent à un code couleur qui vous aidera à les retrouver facilement. Scratch 2 possède plus de 100 blocs, mais certains d'entre eux n'apparaissent que sous certaines conditions. Par exemple, les blocs de la palette *Données* (abordée dans les chapitres 5 et 9) n'apparaissent qu'après la création d'une variable ou d'une liste. Regardons de quoi est composé l'onglet des blocs (figure 1-7).

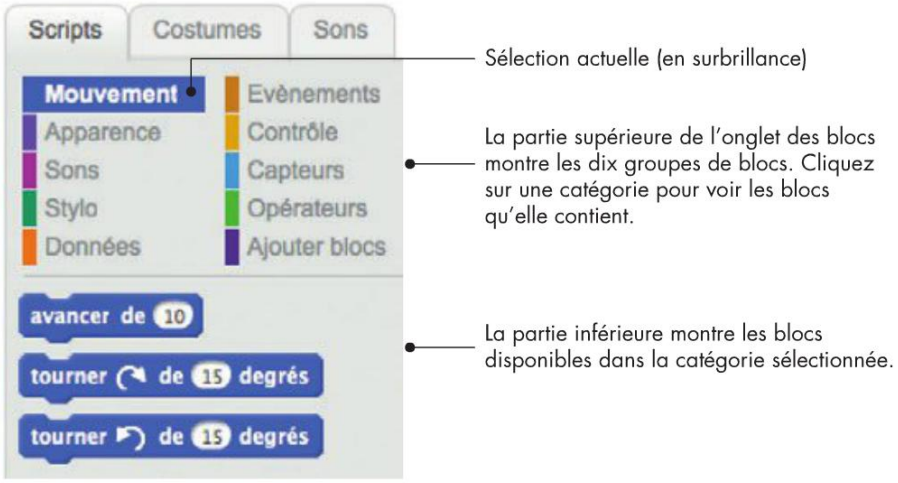


Figure 1-7 : Vue élargie de l'onglet des blocs

Essayez de cliquer sur un bloc pour voir ce qu'il fait. Si vous cliquez sur **avancer de 10** dans la palette *Mouvement*, par exemple, le lutin va avancer de 10 pas sur la scène. Cliquez à nouveau et le lutin avancera encore de

Copyright © 2016 Eyrolles.

10 pas. Cliquez sur le bloc **dire Hello ! pendant 2 secondes** (dans la palette *Apparence*) pour que votre lutin vous salue dans une bulle pendant deux secondes. Cliquez sur l'icône représentant un point d'interrogation (dans la barre d'outils) puis sur un bloc pour accéder à un écran d'aide.

Certains blocs requièrent au moins une entrée (appelée également *argument*) qui précise ce qu'il doit faire. Le nombre 10 du bloc **avancer de 10** est un exemple d'argument. Observez la figure 1-8 pour voir de quelles manières vous pourrez agir sur les entrées des blocs.

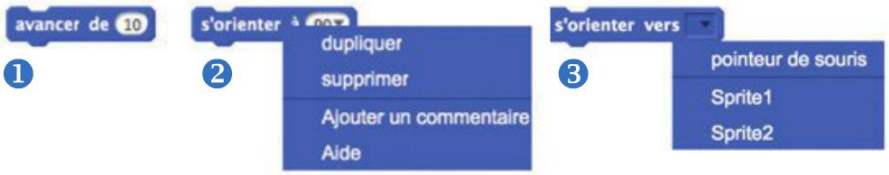


Figure 1-8 : Modification des entrées de différents types de blocs

Vous pouvez changer le nombre de pas dans le bloc **avancer de 10** en cliquant dans la zone blanche et en y entrant un nouveau nombre ❶. Certains blocs, comme **s'orienter à 90**, possèdent un menu déroulant comme entrée ❷. Vous pouvez cliquer sur la flèche pour voir une liste des options disponibles et en sélectionner une. Cette commande spécifique possède une zone blanche éditable, où vous êtes libre d'entrer n'importe quelle valeur. D'autres blocs, comme **s'orienter vers** ❸, vous obligeront à choisir une valeur à partir d'un menu déroulant.

FAITES LE TEST 1-4

Dirigez-vous vers la palette *Apparence*, changez les valeurs des entrées de blocs, puis cliquez sur ces derniers pour voir ce qu'ils font. Réalisez par exemple l'expérience avec le bloc **mettre l'effet couleur à**. Essayez des nombres tels que 10, 20, 30, et ainsi de suite jusqu'à ce que le chat retrouve sa couleur originale. Testez les options du menu déroulant avec différents nombres. Vous pouvez cliquer sur le bloc **annuler les effets graphiques** (également présent dans la palette *Apparence*) pour supprimer vos modifications.

La zone de scripts

Pour faire accomplir des choses intéressantes à votre lutin, vous devez programmer ses actions en faisant glisser les blocs de leur onglet vers la zone de script et en les y assemblant. Lorsque vous les faites glisser dans la zone de scripts, une zone de surbrillance blanche indique un endroit où vous pouvez les déposer pour créer une connexion valide (figure 1-9). Les blocs de Scratch ne s'assemblent que de certaines manières, éliminant les erreurs classiques produites par les personnes qui utilisent des langages de programmation à base de texte.

Copyright © 2016 Eyrolles.

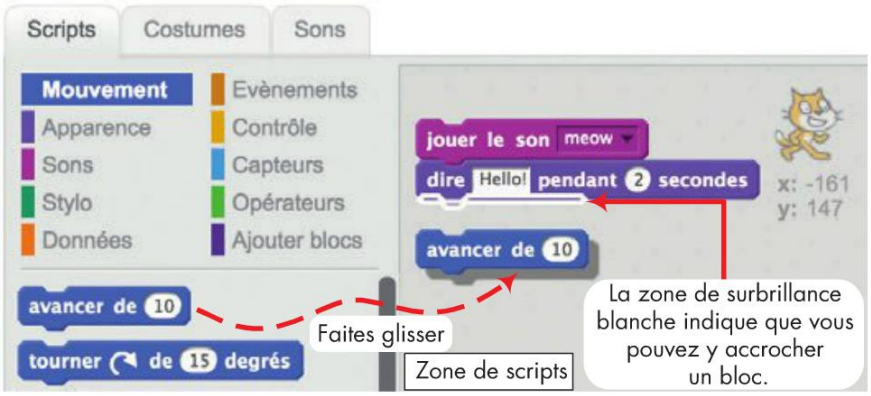
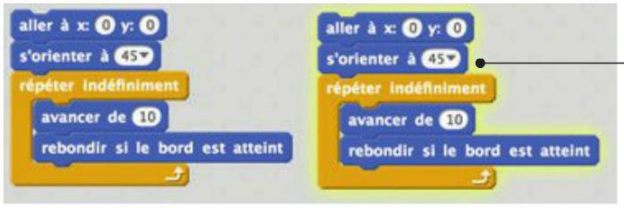


Figure 1-9 : Faites glisser les blocs dans la zone de scripts et assemblez-les pour définir les actions de vos lutins.

Vous n'avez pas besoin de compléter les scripts pour les exécuter, ce qui signifie que vous pouvez les tester pendant leur construction. Cliquer n'importe où sur un script, partiel ou complet, l'exécute entièrement, de haut en bas.

FAITES LE TEST 1-5

Commencez un nouveau projet Scratch et créez le script présenté ci-après pour le lutin Chat (le bloc **répéter indéfiniment** se trouve dans la palette *Contrôle* et les autres dans la palette *Mouvement*).



Vous étudierez la plupart de ces blocs dans le chapitre 2. Pour l'instant, cliquez sur votre nouveau script pour l'exécuter. Scratch devrait mettre en évidence les scripts en cours d'exécution grâce à des contours jaunes, comme sur le côté droit de l'image. Vous pouvez même changer l'entrée d'un bloc et ajouter des blocs à un script en cours d'exécution ! Par exemple, changez le nombre dans le bloc **avancer de** et regardez comment en sont affectés les mouvements du chat. Cliquez une fois de plus sur le script pour l'arrêter.

Vous pouvez également facilement désassembler une pile et tester chaque bloc individuellement. Ce sera une stratégie inestimable lorsque vous essayerez de comprendre un long script. Pour déplacer une pile entière, attrapez le bloc supérieur de la pile. Pour détacher un bloc au milieu d'une pile et tous ceux qui sont en-dessous, attrapez-le et faites-le glisser. Essayez-donc tout ça !

Cette fonction vous permet également de construire votre projet morceau par morceau. Vous construisez de petits assemblages de blocs, les testez pour vous assurer qu'ils fonctionnent comme vous le désirez, puis les combinez avec un script plus grand.

Copyright © 2016 Eyrolles.

Vous pouvez même copier une pile de blocs d'un lutin à un autre. Pour ce faire, il vous suffit de faire glisser la pile à partir de la zone de scripts du lutin source vers la miniature du lutin de destination.

FAITES LE TEST 1-6

Ajoutez un nouveau lutin à votre projet. Faites glisser le script de *Chat* vers la miniature du nouveau lutin. La flèche de votre souris doit se trouver sur la nouvelle miniature pour que l'opération fonctionne. Vérifiez la zone de scripts du nouveau lutin pour vous assurer qu'il contienne bien une copie identique au script original.

L'onglet Costumes

Vous modifiez l'apparence d'un lutin en changeant son costume, qui n'est en fait qu'une image. L'onglet **Costumes** contient tout ce dont vous avez besoin pour organiser les apparences de votre lutin ; voyez-le comme un placard à vêtements ; il peut contenir beaucoup de vêtements, mais un lutin n'en porte qu'un seul à la fois.

Essayons de changer le costume de Chat. Cliquez sur la miniature du lutin et sélectionnez l'onglet **Costumes**. Deux apparences sont disponibles (figure 1-10) : *costume1* (en surbrillance, c'est le costume actuel) et *costume2*.

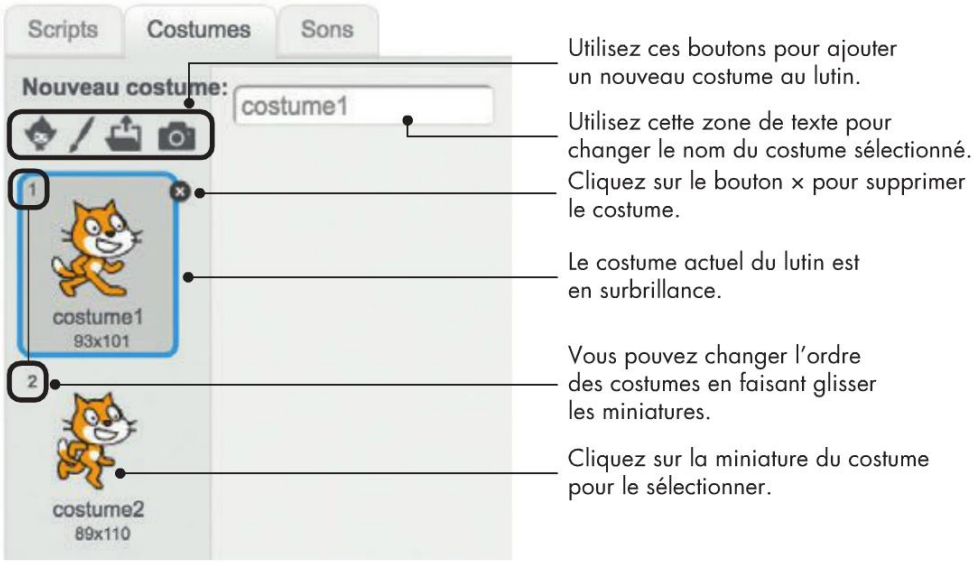


Figure 1-10 : Vous organisez les apparences d'un lutin dans l'onglet Costumes.

Si vous cliquez droit sur la miniature d'un costume, un menu contextuel apparaît avec les options suivantes : **dupliquer**, **supprimer** et **enregistrer localement comme fichier**. La première option crée un nouveau costume ayant la même image. La seconde supprime celui qui est sélectionné. La dernière option le sauvegarde dans un fichier. Vous pourrez importer ce costume et l'utiliser dans d'autres projets en utilisant le bouton **importer le costume depuis un fichier** (le troisième bouton sur la figure 1-10). Essayez toutes ces options.

Copyright © 2016 Eyrolles.

FAITES LE TEST 1-7

Cliquez sur le premier bouton au dessus de l'image du chat sur la figure 1-10 pour choisir un costume à partir de la bibliothèque de Scratch. Sélectionnez alors une image qui vous plaît dans la fenêtre qui apparaît. Appliquez les conseils de la figure 1-10 pour vous familiariser avec les options des costumes.

L'onglet Sons

Les lutins savent également jouer des sons, ce qui rend vos programmes plus vivants. Vous pouvez, par exemple, donner un son différent à un lutin lorsqu'il est triste ou joyeux. Et si votre jeu contient un lutin ayant une apparence de missile, donnez-lui un son différent lorsqu'il atteint ou rate une cible.

Les boutons de l'onglet **Sons** servent à organiser les différents sons que votre lutin peut jouer. Scratch fournit même un outil d'édition de fichiers audio (figure 1-11). Je n'en parlerai pas en détail dans ce livre, mais je vous encourage à l'expérimenter par vous-même.

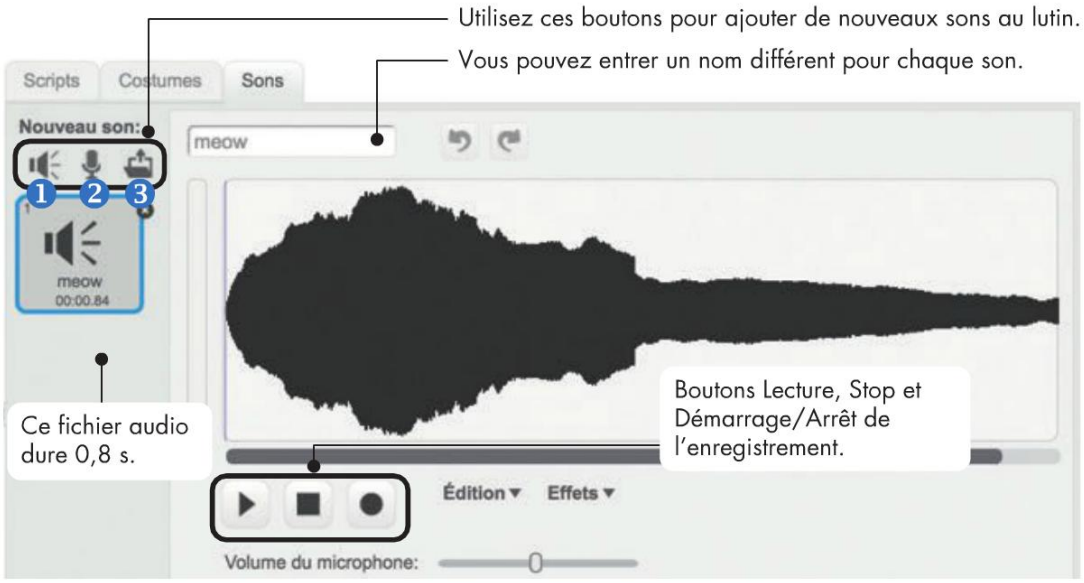


Figure 1-11 : L'onglet Sons sert à organiser les sons d'un lutin.

La plupart du temps, vous n'aurez besoin que des trois boutons en haut de l'onglet **Sons**. Ils permettent de choisir un son à partir de la bibliothèque de Scratch ❶, d'enregistrer un nouveau son ❷ (si vous avez un microphone), ou d'importer un son existant à partir de votre ordinateur ❸. Scratch peut lire uniquement les fichiers audio WAV et MP3.

FAITES LE TEST 1-8

Sélectionnez l'onglet **Sons** et cliquez sur le bouton **choisir un son dans la bibliothèque**. Écoutez les différents sons disponibles dans Scratch pour vous en inspirer dans vos futurs projets.

Copyright © 2016 Eyrolles.

L'onglet Arrière-plans

Quand vous sélectionnez la miniature de la scène dans la liste des lutins, le nom de l'onglet central change et passe de **Costumes** à **Arrière-plans**. Utilisez cet onglet pour organiser les fonds d'écran de la scène, que vous pourrez changer avec vos scripts. Par exemple, si vous créez un jeu, vous pourriez montrer un arrière-plan avec les instructions au début et le changer quand l'utilisateur démarre le jeu. L'onglet **Arrière-plans** est identique à **Costumes**.

FAITES LE TEST 1-9

Cliquez sur le bouton **choisir un arrière-plan dans la bibliothèque** sous la miniature de la scène dans la liste des lutins. Sélectionnez l'image *xy-grid* et faites-en l'arrière-plan par défaut. Cette image montre un plan cartésien 2D, très utile lorsque vous travaillez avec les blocs de commande de **Mouvement**. Répétez ces étapes et sélectionnez un autre arrière-plan de votre choix.

Les informations du lutin

Vous pouvez consulter les informations du lutin en cliquant sur la petite icône ⓘ dans le coin supérieur gauche de sa miniature (figure 1-12). Cette zone montre le nom du lutin, sa position (*x*,*y*) actuelle et sa direction, son style de rotation, son état de visibilité et s'il est possible de le faire glisser ou non en mode Présentation. Discutons brièvement de chacune de ces options.



Figure 1-12 : La zone d'informations du lutin

La zone de texte ❶ vous permet de changer le nom du lutin. Vous l'utiliserez à de nombreuses reprises dans ce livre.

Les valeurs **x** et **y** ❷ montrent la position actuelle du lutin dans la scène. Faites glisser celui-ci et regardez comment ces nombres en sont affectés.

La **direction** ❸ indique dans quel sens le lutin se déplacera en réponse à un bloc de mouvement. Déplacez la ligne bleue émanant du centre du cercle pour faire tourner le lutin.

Les trois boutons de **style de rotation** ❹ (**rotation**, **gauche-droite** et **pas de rotation**) contrôlent la façon dont le costume est affiché lorsque le lutin change de direction. Pour comprendre leur effet, créez le script de la figure 1-13, puis cliquez sur chacun de ces boutons pendant que le script s'exécute. Vous trouverez le bloc **attendre** dans la palette *Contrôle*.

Copyright © 2016 Eyrolles.

La case **peut glisser dans le lecteur** ❺ indique s'il est possible d'insérer le lutin (en utilisant la souris) dans le mode Présentation. Pour comprendre l'effet de cette case, passez en mode Présentation en la cochant/décochant et essayez de faire glisser le lutin dans la scène.

La case **montrer** ❻ montre/cache le lutin au moment de la conception du programme. Essayez-la et observez ce qui se passe. Dans ce livre, de nombreux exemples vous montreront des lutins cachés qui exécutent des tâches utiles en coulisse.

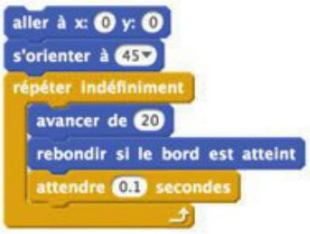


Figure 1-13 : Script affichant les différents styles de rotation

La barre d'outils

Jetons un œil à la barre d'outils de Scratch (figure 1-14), en commençant par les boutons. Une autre barre d'outils est proposée si vous êtes connecté (voir l'annexe). Utilisez les boutons **dupliquer** et **supprimer** pour copier et retirer les lutins, les costumes, les sons, les blocs ou les scripts. Les boutons **agrandir** et **réduire** changent la taille. Cliquez sur le bouton que vous voulez utiliser, puis sur un lutin (ou un script) pour mettre en œuvre son action. Pour remettre votre curseur en forme de flèche, cliquez sur une zone vide de l'écran. Vous pouvez utiliser le menu **langage** pour changer le langage de l'interface utilisateur.

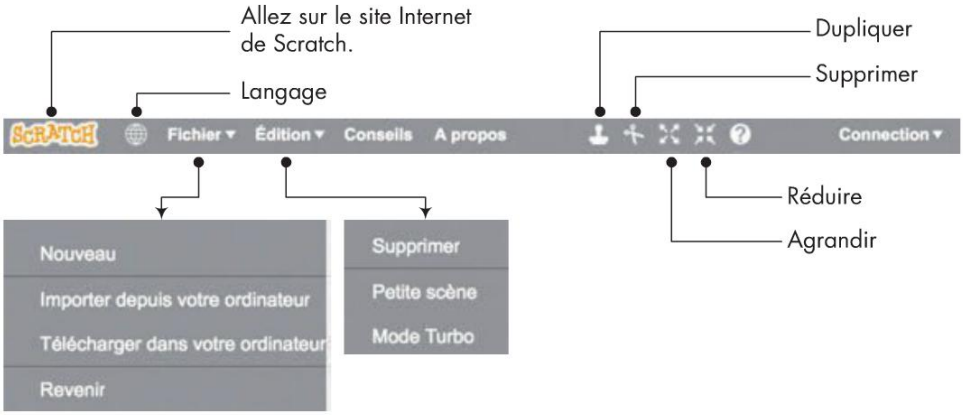


Figure 1-14 : La barre d'outils de Scratch

À partir du menu **Fichier**, vous pouvez créer de nouveaux projets, en importer (ouvrir) un depuis votre ordinateur, télécharger (sauvegarder) le projet actuel sur votre ordinateur, ou revenir à l'état initial (annuler toutes les modifications apportées). Les projets de Scratch 2 sont des fichiers qui ont pour extension *.sb2*, ce qui permet de les distinguer des projets créés dans la version précédente de Scratch (*.sb*).

Dans le menu **Édition**, **Supprimer** annulera la dernière action de suppression effectuée sur un bloc, un script, un lutin, un costume ou un son. L'option **Petite scène** rétrécit la scène et donne plus d'espace à la zone de scripts. La sélection du **Mode Turbo** augmente la vitesse d'exécution de certains blocs. Par exemple, exécuter un bloc **avancer** 1 000 fois peut prendre environ 70 s en mode normal, contre 0,2 s en mode Turbo.

Copyright © 2016 Eyrolles.

Maintenant que vous avez vu l'essentiel de la barre d'outils de Scratch, nous allons discuter brièvement de l'éditeur d'images intégré.

L'éditeur d'images

Vous pouvez utilisez l'éditeur d'images (figure 1-15) pour créer ou éditer des costumes et des arrière-plans (ou tout autre logiciel à votre convenance).

Si vous voulez en savoir plus sur l'éditeur d'images de Scratch, consultez le fichier *ScratchPaintEditor.pdf* (en anglais), situé dans les ressources en ligne, qui peuvent être téléchargées à l'adresse *http://www.editions-eyrolles.com/go/grandlivrescratch*.

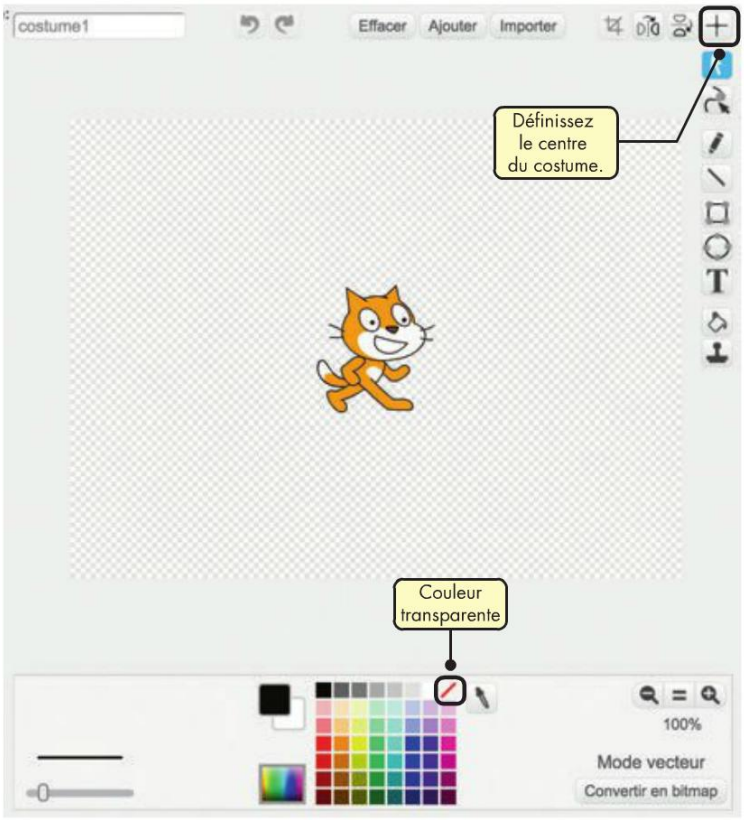


Figure 1-15 : L'éditeur d'images de Scratch

Pour le moment, il y a deux paramètres importants que vous devez connaître : le réglage du centre d'une image et celui d'une couleur transparente.

Définir le centre d'une image

Lorsque vous indiquez à un lutin de tourner (à gauche ou à droite) il le fera par rapport à un point de référence : le centre du costume. Le bouton **définir le centre du costume** (dans le coin supérieur droit de l'éditeur d'images) vous permet d'en choisir l'emplacement. Lorsque vous cliquez sur ce bouton, deux axes noirs apparaissent dans la zone de dessin (figure 1-16). Le point

Copyright © 2016 Eyrolles.

central est défini par leur intersection. Faites glisser ce point d'intersection pour définir la nouvelle position du centre de l'image. Pour cacher ces axes, cliquez sur le même bouton une nouvelle fois.

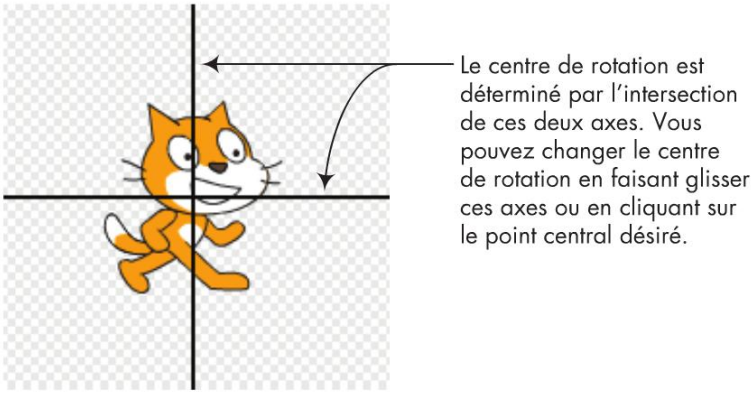
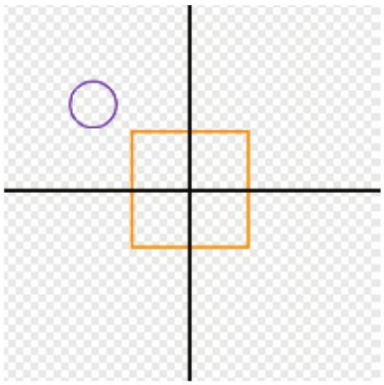


Figure 1-16 : Changement du centre d'un costume après avoir cliqué sur le bouton Définir le centre du costume

CentreDeRotation.sb2

FAITES LE TEST 1-10

Ouvrez *CentreDeRotation.sb2* et exécutez-le. Cette application contient un unique lutin possédant le costume et le script ci-après. Le centre du costume a été défini au centre du carré. Exécutez le script et observez le motif. Ensuite, éditez le costume en plaçant le centre au milieu du cercle, puis réexécutez le script pour voir de quelle manière le motif est affecté.





Définir une couleur transparente

Lorsque deux images se chevauchent, l'image qui est en premier plan couvre une partie de l'image en second plan. De même, les lutins couvrent une partie de la scène. Si vous voulez voir l'allure de cette dernière derrière une image, vous devez utiliser l'éditeur d'images pour rendre au moins une partie de l'image *transparente*, tout comme le pingouin de droite de la figure 1-17.

Dans la palette de couleurs, cliquez sur le carré avec une ligne diagonale rouge et peignez avec cette couleur « transparente » pour rendre une partie de l'image invisible.

Copyright © 2016 Eyrolles.

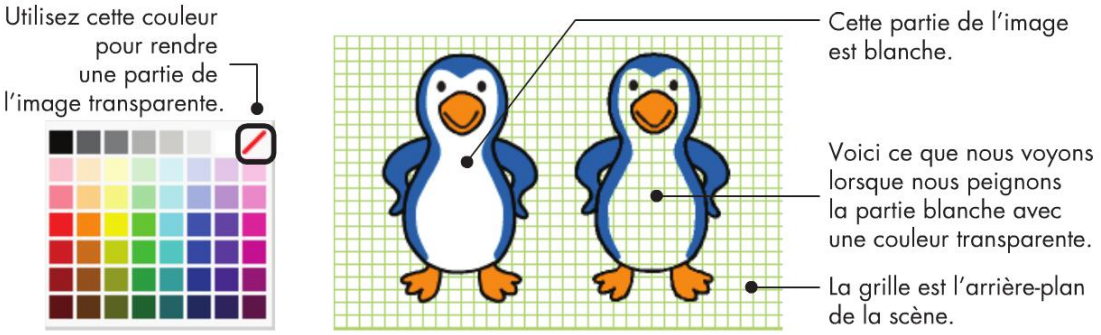


Figure 1-17 : Vous pouvez rendre une partie de l'image transparente en la replissant avec la couleur « transparent ».

Maintenant que vous avez fait le tour de l'interface de Scratch, mettons ces connaissances à profit en faisant quelque chose d'amusant. Retroussez vos manches et préparez-vous : nous allons créer un jeu !

Votre premier jeu Scratch

Pong.sb2
Pong_Pas DeCode.sb2

Dans cette section, vous allez créer un jeu à un joueur, basé sur un grand classique des jeux d'arcade, Pong. Le joueur doit déplacer une raquette pour empêcher une balle rebondissante de toucher le sol. L'interface utilisateur de notre jeu est illustrée à la figure 1-18.

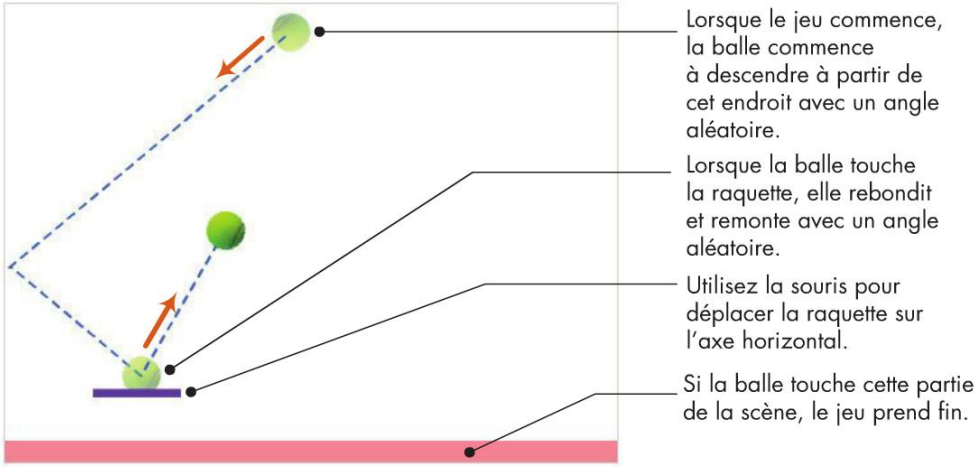


Figure 1-18 : L'écran de notre jeu

Comme le montre la figure, la balle démarre en haut de la scène et se déplace vers le bas avec un angle aléatoire, en rebondissant sur les bords. Le joueur déplace la raquette horizontalement (avec la souris) pour faire remonter la balle. Si la balle touche le bas de la scène, la partie est terminée.

Nous allons développer ce jeu étape par étape, après avoir ouvert un nouveau projet. Cliquez sur **Fichier>Nouveau** pour démarrer un nouveau projet Scratch. Supprimez le lutin Chat en effectuant un clic droit et en sélectionnant **supprimer** du menu contextuel.

Copyright © 2016 Eyrolles.

Étape 1. Préparez l'arrière-plan

Pour détecter le moment où la raquette rate la balle, nous allons attribuer une couleur au bas de la scène et utiliser le bloc **couleur touchée ?** (de la palette *Capteurs*), qui détectera quand la balle touchera cette couleur. Notre arrière-plan actuel est blanc ; nous pouvons donc simplement dessiner un rectangle coloré fin en bas de la scène (figure 1-19).



Figure 1-19 : Les étapes pour dessiner un rectangle en bas de l'image d'arrière-plan

Cliquez sur la miniature de la scène puis sur l'onglet **Arrière-plans**. Suivez les étapes de la figure 1-19 pour dessiner un rectangle fin en bas.

Étape 2. Ajoutez la raquette et la balle

Cliquez sur le bouton **dessiner un nouveau lutin** au-dessus de la liste des lutins pour ajouter Raquette à votre projet. Puisque ce n'est rien d'autre qu'un rectangle fin, répétez les mêmes étapes que précédemment pour dessiner une raquette comme celle de la figure 1-18. Coloriez-la de la couleur de votre choix et définissez son centre au milieu du rectangle.

Ensuite, donnez au lutin un nom qui le représente ; j'ai appelé le mien Raquette. Faites-le glisser sur la scène afin de lui donner une coordonnée **y** d'environ -120.

Il manque encore la balle. Cliquez sur le bouton **choisir un lutin dans la bibliothèque** au dessus de la liste. Dans la fenêtre qui s'affiche, cliquez sur la catégorie **Choses** et sélectionnez l'image **Tennis Ball** pour ajouter ce lutin à votre projet. Renommez-le en Balle.

Avant de commencer à travailler sur les scripts du jeu, sélectionnez **Fichier>Télécharger dans votre ordinateur** pour enregistrer votre travail. Dans la fenêtre qui apparaît, sélectionnez le dossier, nommez le projet *pong.sb2* et cliquez sur **Enregistrer**. Si vous êtes connecté à votre compte, vous pouvez également sauvegarder votre travail sur le *cloud* (sur le serveur de Scratch).

Copyright © 2016 Eyrolles.

Que vous décidiez d'enregistrer votre travail localement (sur votre ordinateur) ou en ligne, assurez-vous de le sauvegarder souvent.

Aves les lutins Raquette et Balle, la scène devrait ressembler à la figure 1-18. Si vous rencontrez des difficultés à ce stade, vous pouvez ouvrir le fichier *Pong_PasDeCode.sb2*, qui contient tout ce que nous venons de créer. Vous ajouterez ensuite les scripts qui feront fonctionner le jeu, mais ne vous souciez pas trop des détails des blocs. Nous les étudierons tous plus tard dans le livre. Pour l'instant, concentrons-nous sur l'apprentissage de la création complète d'un projet.

Étape 3. Démarrez le jeu et déplacez vos lutins

En tant que concepteur du jeu, vous déciderez comment les joueurs peuvent démarrer une nouvelle partie. Par exemple, le jeu peut commencer en appuyant sur une touche, en cliquant sur la scène ou sur un lutin, ou même en agitant ou tapant les mains (si vous disposez d'une webcam). L'icône du drapeau vert (au-dessus de la scène) est une autre option populaire que nous utiliserons ici.

L'idée est simple. Tout script commençant par le bloc de déclenchement **quand le drapeau vert est cliqué** démarre son exécution lorsque vous appuyez sur ce bouton. Le drapeau passera au vert clair et restera ainsi jusqu'à ce que le script se termine. Pour le voir en action, créez le script présenté à la figure 1-20 pour le lutin Raquette.

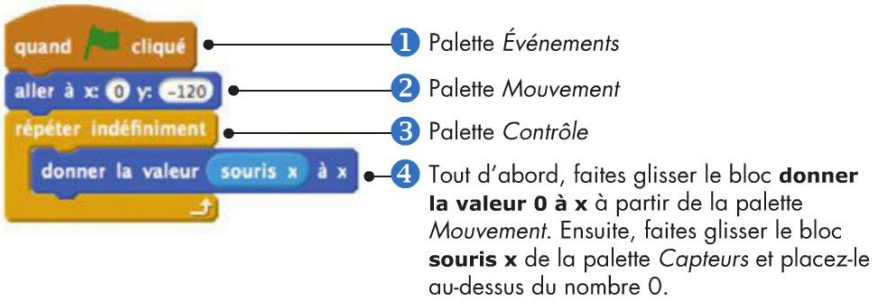


Figure 1-20 : Le script du lutin Raquette

Quand on clique sur le drapeau vert ❶, le bloc **aller à x : y :** ❷ définit la position verticale de la raquette à -120, au cas où vous l'auriez préalablement déplacée avec la souris. La raquette devrait se trouver juste au-dessus du rectangle rose du bas de la scène. Si votre rectangle est trop épais, changez la position de la raquette afin de l'adapter.

Ensuite, le script utilise le bloc **répéter indéfiniment** ❸ afin de vérifier constamment la position de la souris. Nous déplacerons la raquette de gauche à droite en faisant coïncider sa position avec celle de la souris ❹. Exécutez le script (en cliquant sur le drapeau vert) et essayez de déplacer votre souris sur l'axe horizontal. La raquette devrait la suivre. Cliquez sur l'icône de stop à droite du drapeau vert pour arrêter le script.

Le script du lutin Balle est légèrement plus long que le précédent. La balle doit commencer à se déplacer quand on clique sur le drapeau vert. Ajoutez donc d'abord le script de la figure 1-21 au lutin Balle.

Copyright © 2016 Eyrolles.

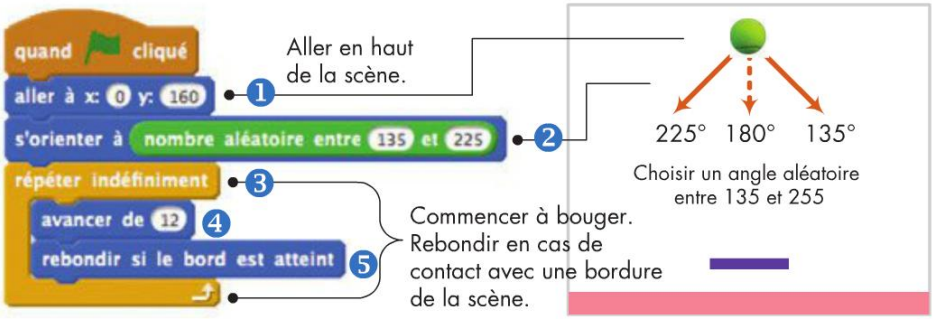


Figure 1-21 : La première partie du script du lutin Balle

Tout d'abord, nous plaçons la balle en haut de la scène ❶ et lui indiquons de s'orienter selon un angle aléatoire grâce au bloc **nombre aléatoire** ❷ (de la palette *Opérateurs*). Le script utilise ensuite le bloc **répéter indéfiniment** ❸ pour déplacer la balle ❹ à travers la scène et la faire rebondir ❺ sur les bords. Cliquez sur le bouton vert pour tester ce que vous avez écrit jusque-là. La balle devrait se déplacer en zigzag et la raquette devrait toujours suivre votre souris.

FAITES LE TEST 1-11

Remplacez le 12 du bloc **avancer de** par différentes valeurs, exécutez le script et observez le résultat. Cela devrait vous indiquer comment rendre un jeu plus simple ou plus difficile à jouer. Cliquez sur l'icône **stop** lorsque vous aurez terminé.

Il est temps à présent d'ajouter la partie amusante : les blocs qui feront rebondir la balle sur la raquette. Nous pouvons modifier l'intérieur du bloc **répéter indéfiniment** que nous venons de créer pour que la balle s'oriente vers le haut lorsqu'elle entre en contact avec la raquette (figure 1-22).

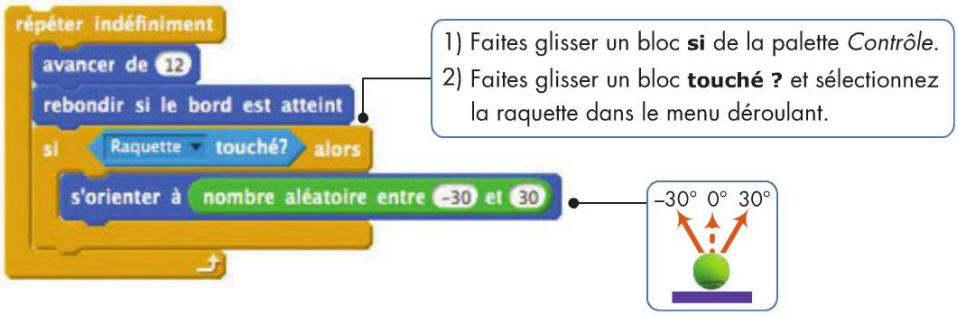


Figure 1-22 : Ajout du code qui fait remonter la balle

Quand la balle touche la raquette, nous lui indiquons de s'orienter dans une direction aléatoire comprise entre -30 et 30. Quand le bloc **répéter indéfiniment** passe au cycle suivant, il exécute le bloc **avancer de**, qui déplace la balle vers le haut. Cliquez à nouveau sur le drapeau vert pour tester cette partie du jeu. Cliquez sur l'icône **stop** lorsque vous vous serez assuré que la balle rebondit bien sur la raquette.

Copyright © 2016 Eyrolles.

À présent, la seule partie manquante est le code qui arrêtera le jeu lorsque la balle touchera le bas de la scène. Ajoutez le script de la figure 1-23 au lutin Balle, soit juste avant soit juste après le bloc **si** de la figure 1-22. Vous trouverez le bloc **couleur touchée ?** dans la palette *Capteurs* et le bloc **stop** dans la palette *Contrôle*.



Figure 1-23 : Les blocs qui mettent fin à la partie

Lorsque vous cliquez sur le carré de couleur dans le bloc **couleur touchée ?**, le curseur se transforme en main. Déplacez-la et cliquez sur le rectangle rose en bas de la scène ; le carré du bloc prend la couleur du rectangle. Le bloc **stop tout** se comporte exactement comme son nom l'indique : il arrête tous les scripts en cours d'exécution de tous les lutins, sans exception.

Ce jeu de Pong basique est à présent fonctionnel. Cliquez sur le drapeau vert et jouez-y une ou deux fois pour le tester. Maintenant que vous voyez la si faible quantité de code nécessaire à la création d'un jeu vidéo complet, j'espère bien que vous rejoignez mon avis lorsque j'affirme que Scratch est génial !

Étape 4. Mettez de l'ambiance avec du son

Les jeux sont évidemment plus amusants lorsqu'ils contiennent du son. Ajoutons donc une touche finale en jouant un bruit à chaque fois que nous touchons la balle.

Double-cliquez sur la balle dans la scène pour la sélectionner et ouvrez l'onglet **Sons**. Cliquez sur le bouton **choisir un son dans la bibliothèque**. Dans la fenêtre qui apparaît, sélectionnez la catégorie **Effets**, choisissez le son **pop** et cliquez sur **OK** pour l'ajouter à l'onglet **Sons**. Retournez ensuite à l'onglet **Scripts** et ajoutez-y le bloc **jouer le son** de la palette *Sons* (figure 1-24).



Figure 1-24 : Jouez un son lorsque la balle touche la raquette.

Testez le jeu une fois de plus. Vous devriez cette fois-ci entendre un court « pop » chaque fois que la balle touche la raquette.

Félicitations ! Votre jeu est à présent complet (à moins, bien sûr, que vous ne désiriez y ajouter des fonctions) et vous venez d'écrire votre premier programme Scratch. Si vous désirez aller plus loin, essayez de dupliquer le lutin Balle pour obtenir deux balles (ou plus) dans le jeu et voir comment cela affecte la manière de jouer !

Copyright © 2016 Eyrolles.

Dans la section suivante, nous présenterons les différents types de blocs disponibles dans Scratch. Au cours de votre lecture, vous étudierez en détail la façon dont ces blocs fonctionnent mais, pour l'instant, nous ne les présenterons que brièvement.

Les blocs de Scratch : un aperçu

Dans cette section, vous prendrez connaissance des différents blocs de Scratch, de leurs noms et de leur rôle. Le but est de définir certains termes que vous lirez dans les chapitres suivants. Vous pourrez revenir à cette section à tout moment si vous souhaitez vous rafraîchir la mémoire.

Scratch possède quatre types de blocs (figure 1-25) : les commandes, les fonctions, les déclenchements et les contrôles. Les *blocs de commande* et les *blocs de contrôle* (également appelés *blocs d'empilement*) possèdent une languette en bas et/ou une encoche en haut. Vous pouvez les assembler pour former des piles. Les *blocs de déclenchement*, également appelés *chapeaux*, ont un bord supérieur arrondi car ils se placent en haut d'une pile. Ils attendent un événement – tel qu'un clic ou un appui sur une touche – et exécutent les blocs au-dessus desquels ils se trouvent lorsque l'événement se produit. Par exemple, les scripts commençant par un bloc **quand le drapeau vert est cliqué** s'exécuteront lorsque l'utilisateur cliquera sur l'icône du drapeau vert.

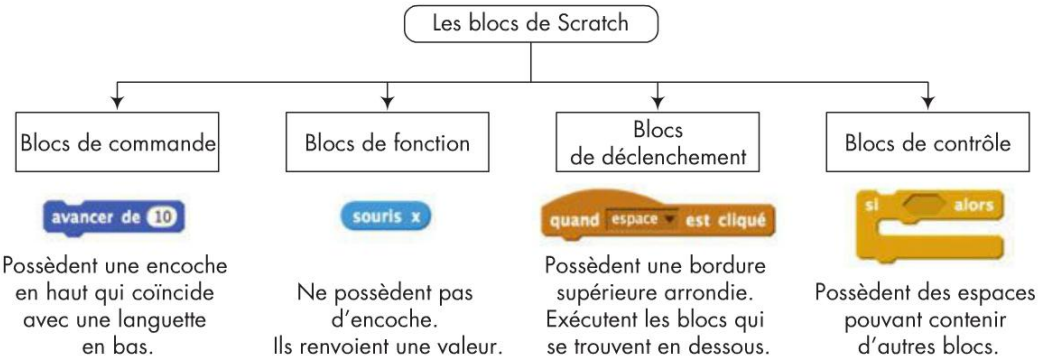


Figure 1-25 : Les quatre types de blocs disponibles dans Scratch

Les *blocs de fonction* (aussi appelés *rapporteurs*) n'ont ni encoche ni languette. À eux seuls, ils ne peuvent pas constituer une couche d'un script ; ils sont plutôt utilisés comme entrées pour d'autres blocs. Leur forme indique le type de données qu'ils renvoient. Par exemple, les blocs aux côtés arrondis renvoient des nombres ou des suites de caractères, alors que les blocs aux côtés pointus indiquent si quelque chose est vrai ou faux (figure 1-26).

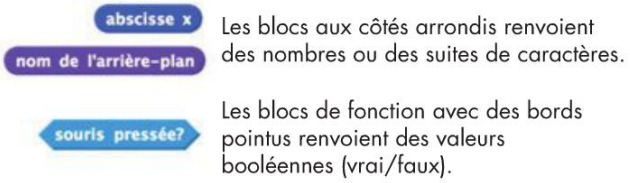


Figure 1-26 : La forme d'un bloc de fonction indique le type de données qu'il renvoie.

Copyright © 2016 Eyrolles.

Certains blocs de fonction sont accompagnés d'une case à cocher. Si vous cochez cette dernière, un *moniteur* apparaît sur la scène pour afficher la valeur actuelle du rapporteur. Sélectionnez un lutin et cochez la case accompagnant le bloc **abscisse x** (dans la palette *Mouvement*). Faites ensuite glisser le lutin dans la scène et observez ce moniteur. Sa valeur devrait changer lorsque vous déplacez le lutin de gauche à droite.

Opérateurs arithmétiques et fonctions

Regardons à présent les opérateurs arithmétiques et les fonctions proposés par Scratch. Si vous ne retrouvez plus votre calculatrice, alors vos soucis sont terminés ! Vous pourriez créer votre propre calculatrice dans Scratch avec des blocs de la palette *Opérateurs*, que nous allons explorer dans cette section.

Opérateurs arithmétiques

Scratch fournit les opérations arithmétiques basiques d'addition (+), de soustraction (-), de multiplication (*) et de division (/). Les blocs utilisés pour effectuer ces opérations, appelés *opérateurs*, sont présentés sur la figure 1-27. Puisqu'ils produisent un nombre, vous pouvez les utiliser comme entrées de tout bloc acceptant les nombres.

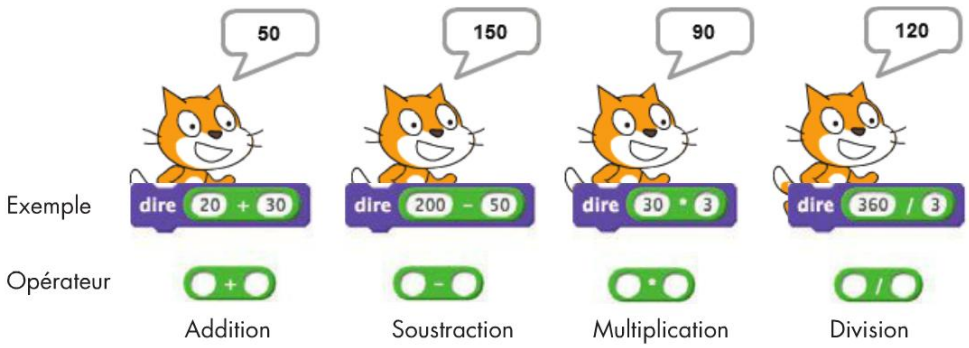


Figure 1-27 : Opérateurs arithmétiques de Scratch

Scratch fournit également l'opérateur modulo, qui renvoie le reste d'une division. Par exemple, **10 modulo 3** renvoie 1 car le reste d'une division de 10 par 3 est 1. Cet opérateur est souvent utilisé pour tester la divisibilité d'un *entier* par un autre (plus petit). Un modulo de 0 indique que le plus grand nombre est divisible par le plus petit. Cela vous aide-t-il à deviner comment vérifier si un nombre est pair ou impair ?

Un autre opérateur utile que fournit Scratch est l'**arrondi**, qui approche un nombre décimal par le nombre entier le plus proche. Par exemple, **arrondi de (3.1)** donne 3, **arrondi de (3.5)** donne 4 et **arrondi de (3.6)** donne 4.

Nombres aléatoires

En programmant, vous aurez probablement besoin de générer des nombres tirés aléatoirement, surtout si vous créez des jeux ou des simulations. Scratch fournit le bloc **nombre aléatoire** pour ce faire.

Copyright © 2016 Eyrolles.

Ce bloc renvoie un nombre aléatoire à chaque fois que vous l'utilisez. Ses deux zones de texte servent à définir une plage pour ce nombre et Scratch ne choisira que des valeurs comprises entre ces deux limites (inclusives). La figure 1-28 montre des exemples d'utilisation de ces blocs.

Tableau 1-1 : Exemples d'utilisation du bloc **nombre aléatoire**

Exemple	Sorties possibles
	{0, 1}
	{0, 1, 2, 3, … , 10}
	{–2, –1, 0, 1, 2}
	{0, 10, 20,30, …, 100}
	{0,0, 0,1, 0,15, 0,267, 0,3894, …, 1,0}
	{0, 0,01, 0,12, 0,34, 0,58, … , 1,0}

NOTE *Les sorties de* **nombre aléatoire entre 0 et 1** *et* **nombre aléatoire entre 0 et 1.0** *sont différentes. Le premier renvoie soit 1 soit 0, alors que le second renvoie une valeur décimale comprise entre 0 et 1. Si l'une des entrées du bloc* **nombre aléatoire** *contient un point décimal, alors la sortie sera également une valeur décimale.*

Fonctions mathématiques

Scratch met à disposition un grand nombre de fonctions mathématiques. Le bloc **racine de** en regroupe 14 qui peuvent être sélectionnées à partir du menu déroulant, comprenant la racine carrée, les fonctions trigonométriques, logarithmiques et exponentielles. Référez-vous au document (en anglais) *MathematicalFunctions.pdf* des ressources web du livre pour de plus amples informations sur ces fonctions.

Résumé

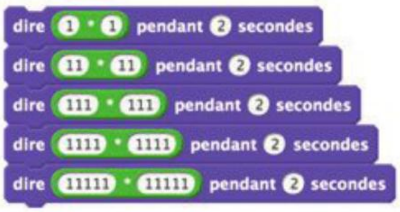
Ce chapitre a fourni un aperçu global de Scratch et de son environnement de programmation. Vous avez pris connaissance des différents éléments de l'interface utilisateur et vous avez même créé un jeu ! Nous avons également exploré les opérateurs et fonctions mathématiques de Scratch.

Vous avez reçu à ce stade la plupart des informations basiques dont vous avez besoin pour créer des scripts puissants dans Scratch, mais ce n'est qu'un petit pas sur le chemin qui vous mènera vers l'écriture de programmes géniaux. Dans le chapitre suivant, vous verrez plus en détail la façon dont Scratch vous aidera à développer vos capacités de programmeur.

Copyright © 2016 Eyrolles.

Problèmes

1. Écrivez le résultat de chaque bloc du script ci-après. Y a-t-il une logique dans le résultat de ces produits ?



2. Y a-t-il une logique dans le résultat des produits 9 × 9, 99 × 99, 999 × 999, etc. ?
3. Complétez le tableau suivant en écrivant les valeurs de chaque expression. Ensuite, utilisez la commande **dire** et les blocs opérateurs appropriés pour vérifier vos réponses.

Expression	Valeur
3 + (2 × 5)	
(10 / 2) – 3	
7 + (8 × 2) – 4	
(2 + 3) × 4	
5 + (2 × (7 – 4))	
(11 – 5) × (2 + 1) / 2	
5 × (5 + 4) – 2 × (1 + 3)	
(6 + 12) mod 4	
3 × (13 mod 3)	
5 + (17 mod 5) – 3	

4. Calculez, sur papier, les expressions Scratch suivantes, si x = 2 et y = 4.

a) 

b) 

c) 

d) 

e) 

Copyright © 2016 Eyrolles.

5. Utilisez la commande **dire** et les blocs de la palette *Opérateurs* appropriés pour calculer :
   a. la racine carrée de 32 ;
   b. le sinus de 30° ;
   c. le cosinus de 60° ;
   d. l'arrondi de 99,459.
6. Créez un bloc de fonction qui calcule la moyenne de trois nombres : 90, 95 et 98. Affichez le résultat en utilisant le bloc **dire**.
7. Créez un bloc de fonction qui convertit 60 °Fahrenheit en Celsius. Indice : $C = (5/9) \times (F–32)$.
8. Créez un bloc de fonction qui calcule l'aire d'un trapèze de hauteur 4/6 cm et dont les bases mesurent 5/9 cm et 22/9 cm. Indice : $A = 0,5 \times (b1 + b2) \times h$, avec h la hauteur, $b1$ et $b2$ les longueurs des deux bases.
9. Créez un bloc de fonction qui calcule la force nécessaire à une accélération de 3 m/s^2 pour une voiture de 2 000 kg. Indice : *Force = masse × accélération*.
10. Le prix de l'électricité est de 0,06 € par kilowattheure. Créez un bloc de fonction qui calcule le coût de l'utilisation d'un climatiseur de 1 500 watt pendant 2 heures. Indice : *Énergie = puissance × temps*.
11. Grâce à une simple astuce mathématique, vous pouvez utiliser l'opérateur arrondi pour arrondir à la décimale désirée (par exemple, le nombre 5,3567 au dixième près) en trois étapes.
    a. 5,3567 × 10 = 53,567 : multipliez le nombre par 10.
    b. arrondi(53,567) = 54 : arrondissez le résultat.
    c. 54/10 = 5,4 : divisez le résultat précédent par 10.

Comment modifier cette procédure pour arrondir au centième près ? Créez un bloc de fonction qui arrondit 5,3567 au dixième (ou au centième) près et affichez la valeur qu'il renvoie avec un bloc **dire**.

Copyright © 2016 Eyrolles.

2

MOUVEMENTS ET DESSINS

Maintenant que vous êtes familiarisé avec l'interface de Scratch, vous voici prêt à manipuler certains de ses outils de programmation. Dans ce chapitre, vous allez :

- tester les commandes de mouvement et de stylo de Scratch ;
- animer des lutins et les faire bouger dans la scène ;
- dessiner des motifs artistiques ou géométriques et créer des jeux ;
- comprendre l'utilité de l'outil de clonage des lutins.

Il est temps que vous fassiez preuve de créativité en plongeant dans le monde de l'infographie !

Utilisation des commandes de mouvement

Si vous désirez créer des jeux ou d'autres programmes animés, vous aurez besoin d'utiliser les blocs de la palette *Mouvement* pour déplacer les lutins. Ils vous permettront d'indiquer à un lutin de se déplacer à un certain endroit de la scène ou de s'orienter vers une certaine direction.

Copyright © 2016 Eyrolles.

Mouvement absolu

Souvenez-vous : comme nous l'avons vu sur la figure 1-4, la scène est semblable à une grille rectangulaire de 480 × 360 pas, dont le point central est (0,0). Scratch possède quatre commandes de *mouvement absolu* (**aller à**, **glisser à**, **donner la valeur à x** et **donner la valeur à y**) qui indiquent au lutin où aller exactement sur la grille.

NOTE *Si vous souhaitez en savoir plus sur ces blocs, utilisez la fenêtre d'aide de Scratch sur la droite de la zone de scripts. Si vous ne voyez pas cette fenêtre, cliquez sur le point d'interrogation qui se trouve près du coin supérieur droit de l'éditeur de projets de Scratch.*

Pour expliquer comment fonctionnent ces commandes, imaginons que vous vouliez que le lutin Fusée de la figure 2-1 atteigne le lutin Cible en forme d'étoile, situé à la position (200,150). Pour ce faire, la façon la plus évidente est d'utiliser le bloc **aller à**. La coordonnée *x* indique au lutin la position à laquelle il devra se placer sur l'axe horizontal de la scène, alors que la coordonnée *y* lui indique où se placer sur l'axe vertical.

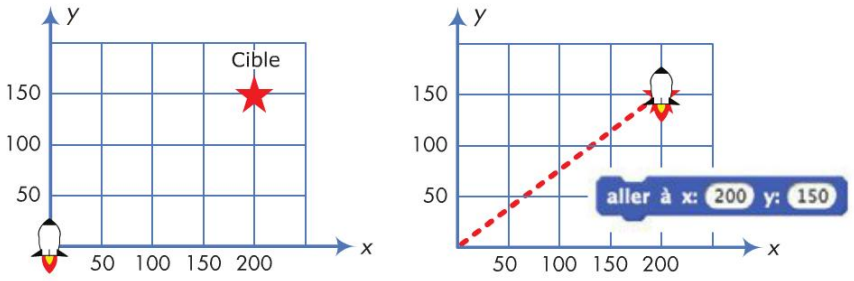


Figure 2-1 : Vous pouvez déplacer un lutin à n'importe quel endroit de la scène en utilisant le bloc **aller à**.

La fusée ne s'orientera pas vers la cible, mais se déplacera sur une ligne invisible connectant la position actuelle, point (0,0), au point (200,150). Vous pouvez faire en sorte que la fusée se déplace plus lentement en utilisant la commande **glisser à**. Elle est presque identique à la commande **aller à**, mais elle vous permet en plus de définir le temps que la fusée prendra pour attendre la cible.

Une autre manière d'attendre la cible est de changer les coordonnées *x* et *y* du lutin Fusée indépendamment en utilisant les blocs **donner la valeur à x** et **donner la valeur à y**, comme le montre la figure 2-2. Vous souvenez-vous du bloc **donner la valeur à x** que vous avez utilisé dans le jeu Pong du chapitre 1 ? (Voir la figure 1-20 page 17 pour réviser.)

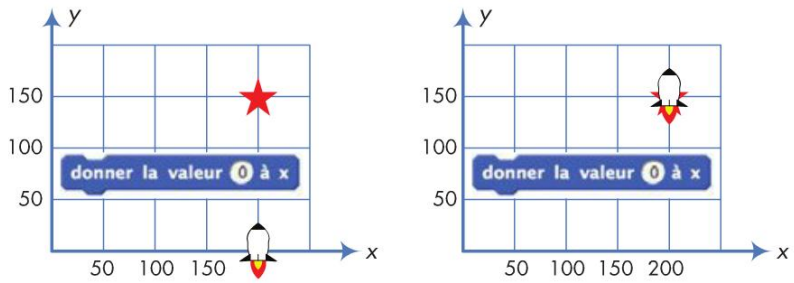


Figure 2-2 : Vous pouvez définir les coordonnées x et y d'un lutin indépendamment l'une de l'autre.

Copyright © 2016 Eyrolles.

Vous pouvez toujours voir la position *(x,y)* d'un lutin dans le coin supérieur droit de la zone de scripts. Si vous voulez afficher ces informations sur la scène, servez-vous des blocs rapporteurs **abscisse x** et **ordonnée y**. Cliquez sur les cases qui les accompagnent pour les cocher et vous verrez leurs valeurs sur la scène.

NOTE *Les commandes de mouvement fonctionnent avec le centre d'un lutin comme référence, que vous pouvez définir dans l'éditeur d'images. Par exemple, le fait d'envoyer un lutin au point (100,100) le déplace de manière à ce que son centre se trouve à (100,100), comme le montre la figure 2-3. Ainsi, lorsque vous dessinez ou importez un costume pour un lutin que vous comptez faire bouger, portez une attention particulière à son centre !*

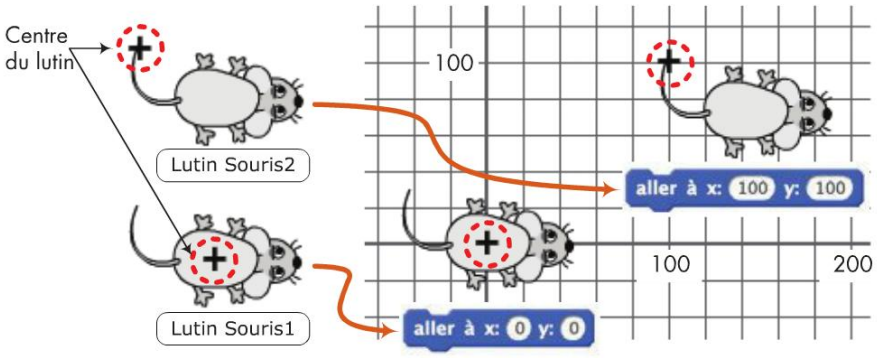
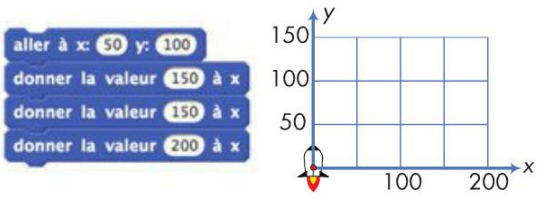


Figure 2-3 : Les commandes de mouvement font référence au centre des lutins.

FAITES LE TEST 2-1

Dressez la liste des coordonnées que prendra le lutin Fusée après avoir exécuté chacune des commandes du script ci-après.



Mouvement relatif

À présent, considérez la grille de la figure 2-4, qui contient un autre lutin Fusée et un autre lutin Cible. Cette fois, vous ne pouvez pas voir les coordonnées ; vous ne connaissez donc pas la position exacte du lutin. Si vous deviez indiquez à la fusée comment atteindre la cible, vous pourriez lui dire : « Avance de trois pas, puis oriente-toi vers la droite, puis avance de deux pas. »

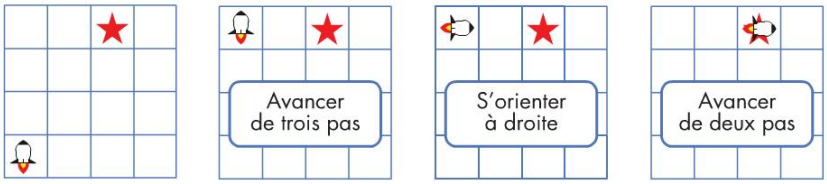


Figure 2-4 : Vous pouvez déplacer un lutin sur la scène en utilisant les commandes de mouvement relatif.

Copyright © 2016 Eyrolles.

Les commandes telles que **avancer de** et **s'orienter à** sont des commandes de *mouvement relatif*. Par exemple, la première commande **avancer de** a fait avancer la fusée vers le haut, alors que la seconde l'a fait avancer vers la droite. La direction du mouvement dépend de (ou est relative à) *l'orientation* actuelle du lutin. La convention d'orientation utilisée dans Scratch est illustrée à la figure 2-5.

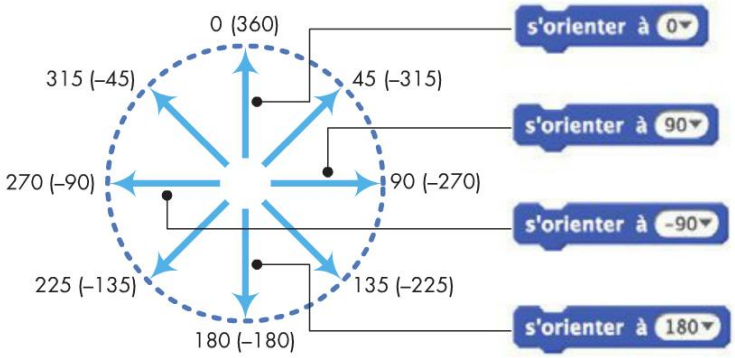


Figure 2-5 : Dans Scratch, 0 pointe vers le haut, 90 vers la droite, 180 vers le bas et -90 vers la gauche.

Vous pouvez tourner un lutin dans une certaine direction (ou *cap*) avec la commande **s'orienter à**. Pour sélectionner le haut, le bas, la droite ou la gauche, cliquez sur la flèche noire et choisissez l'une de ces options dans le menu déroulant. Pour les autres directions, entrez la valeur que vous désirez dans la zone de texte blanche. Vous pouvez même utiliser des valeurs négatives ! Par exemple, les nombres 45 et -315 orienteront tous deux le lutin vers le nord-est.

NOTE *Vous trouverez la direction actuelle du lutin dans sa zone d'information. Vous pouvez également cocher la case qui accompagne le bloc* **direction** *(dans la palette* Mouvement*) pour afficher sa valeur sur la scène.*

Voyons maintenant comment fonctionnent les commandes de mouvement relatif. Nous commencerons par les commandes **avancer de** et **tourner de**, qui fonctionnent selon la direction actuelle du lutin (figure 2-6).

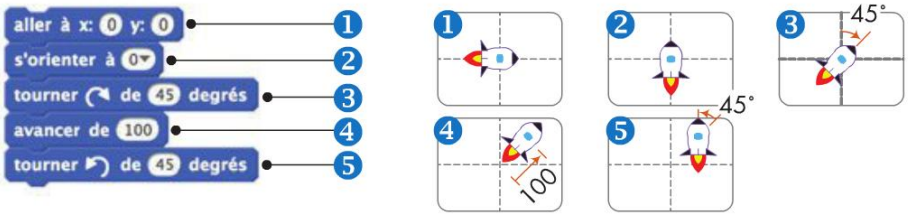


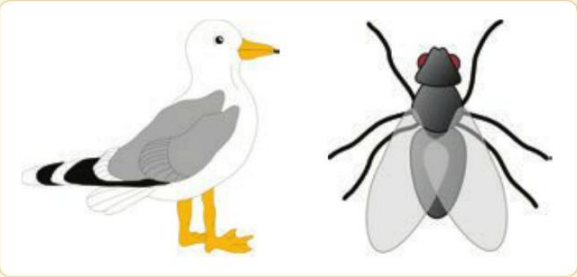
Figure 2-6 : Un script simple qui illustre l'utilisation des commandes **avancer de** *et* **tourner de**

Tout d'abord, le bloc **aller à** ❶ déplace la fusée au centre de la scène. Le deuxième bloc de commande ❷ indique au lutin de s'orienter vers le haut et le troisième ❸ le fait tourner de 45° dans le sens des aiguilles d'une montre. Ensuite, le lutin avance de 100 pas ❹ dans sa direction actuelle avant de tourner une nouvelle fois de 45° dans le sens inverse des aiguilles d'une montre ❺ pour s'arrêter en regardant vers le haut.

Copyright © 2016 Eyrolles.

LA DIRECTION ET LES COSTUMES

La commande **s'orienter à** ne sait rien du costume du lutin. Par exemple, considérez les lutins présentés ci-après.



En utilisant l'éditeur d'images, nous avons dessiné le costume de l'oiseau regardant à droite et celui de l'insecte regardant en haut. Que se passera-t-il si vous utilisez la commande **s'orienter à 90** (qui signifie s'orienter à droite) sur chaque lutin ?

En réalité, aucun des deux ne tournera. Bien que 90° corresponde à « droite », cette direction fait en fait référence à l'*orientation originale* du costume dans l'éditeur d'images. Donc, puisque l'insecte a été dessiné regardant vers le haut, il regardera toujours vers le haut quand vous lui indiquerez de s'orienter à 90°. re lutin réponde à la commande **s'orienter à** comme le montre la figure 2-5, vous devrez le dessiner de façon à ce qu'il regarde vers la droite (tout comme le costume de l'oiseau).

Parfois, vous pourriez n'avoir envie de déplacer votre lutin qu'horizontalement ou verticalement. Les commandes **ajouter à x** et **ajouter à y** entrent alors en jeu. Le script de la figure 2-7 illustre l'utilisation de ces blocs.

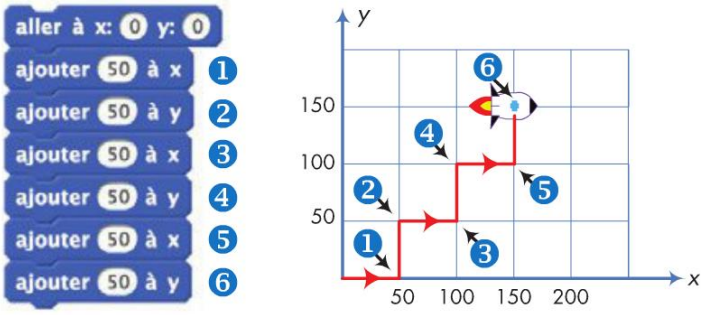


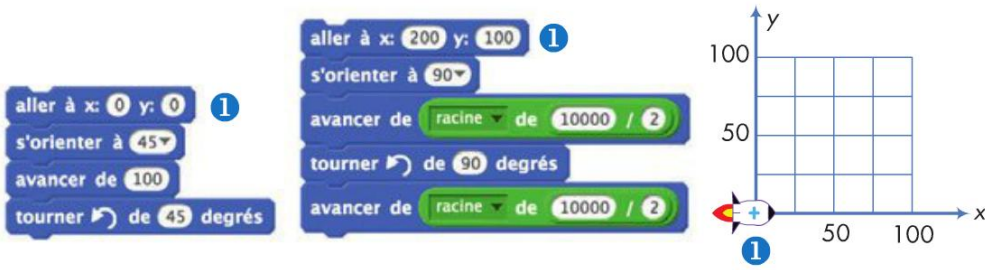
Figure 2-7 : Empruntez un chemin sinueux avec **ajouter à x** *et* **ajouter à y**.

Après avoir atteint le centre de la scène, le lutin Fusée se voit ajouter 50 à sa coordonnée *x* par la commande ❶, ce qui le déplace de 50 pas vers la droite. La commande suivante ❷ fait prendre la valeur 50 à la coordonnée *y*, ce qui déplace le lutin de 50 pas vers le haut. Les autres commandes fonctionnent de la même manière. Le tracé de la figure 2-7 montre le chemin emprunté par le lutin Fusée pour atteindre sa destination finale.

Copyright © 2016 Eyrolles.

FAITES LE TEST 2-2

Trouvez la position *(x,y)* finale de la fusée pour chacun des deux scripts présentés ci-après. Quel théorème mathématique pouvez-vous utiliser pour prouver que ces deux scripts sont équivalents ?



Autres commandes de mouvement

ChasseLaBalle.sb2

Il reste quatre commandes de mouvement à explorer : **s'orienter vers**, un deuxième type de bloc **aller à**, **rebondir si le bord est atteint** et **fixer le sens de rotation**.

Vous avez déjà fait connaissance avec les styles de rotation et vous avez vu la commande **rebondir si le bord est atteint** en action dans le chapitre 1 (figure 1-13). Pour voir l'effet des deux autres commandes, créons l'application simple d'un chat poursuivant une balle (figure 2-8).

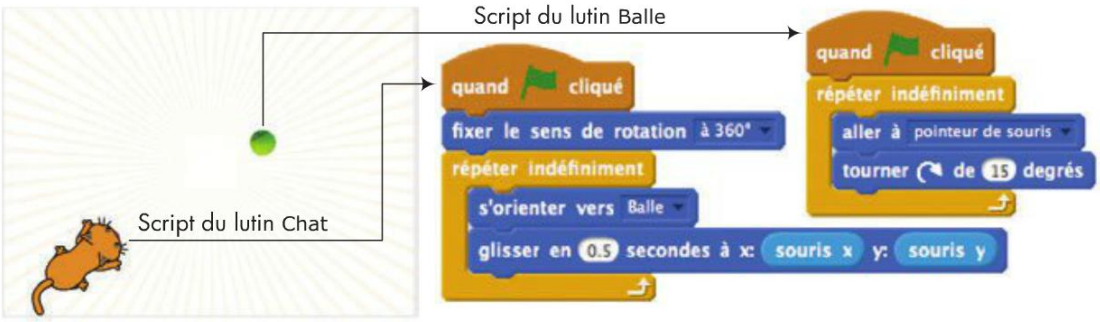


Figure 2-8 : Programmer un chat qui court après une balle de tennis

Cette application contient deux lutins, appelés Chat et Balle. Lorsque vous cliquez sur l'icône du drapeau vert, le lutin Balle suit le pointeur de la souris. Le lutin Chat s'oriente constamment vers la balle et s'en rapproche avec la commande **glisser**. Construisez cette application pour voir comment elle fonctionne. Vous trouverez le bloc **répéter indéfiniment** dans la palette *Contrôle* et les blocs **souris x** et **souris y** dans la palette *Capteurs*. L'application complète est disponible dans le fichier *ChasseLaBalle.sb2*.

Dans la section suivante, nous regarderons la palette *Stylo* et apprendrons comment faire en sorte qu'un lutin laisse une trace de ses mouvements.

Copyright © 2016 Eyrolles.

Dessiner avec les commandes de stylo

DessinFacile.sb2

Les commandes de mouvement que vous avez utilisées dans la section précédente permettent de déplacer les lutins n'importe où sur la scène. Si vous voulez garder la trace des chemins qu'ils empruntent, le stylo de Scratch pourra vous servir !

Chaque lutin possède un *stylo* invisible, qui peut être en position d'écriture ou relevé. Si le stylo est en position d'écriture, le lutin dessinera ses mouvements. S'il est relevé, le lutin se déplacera sans laisser de trace. Les commandes de la palette *Stylo* contrôlent la taille du stylo, sa couleur et son intensité.

FAITES LE TEST 2-3

Ouvrez la fenêtre **Aide** de Scratch, cliquez sur l'icône en forme de maison et cliquez sur **Stylo** dans l'onglet **Blocs** pour obtenir une brève description de chacune des commandes concernées. Les scripts ci-après présentent la plupart d'entre elles. Recopiez ces scripts, exécutez-les et décrivez le résultat que chacun produit. N'oubliez pas de régler le stylo du lutin en position d'écriture avant de les exécuter. Vous trouverez le bloc **répéter** dans la palette *Contrôle*.



Explorons quelques commandes de stylo en détail et créons un programme simple dessinant des images en déplaçant un lutin dans la scène avec les flèches directionnelles. Un appui sur la flèche du haut (↑) fera avancer le lutin de 10 pas. Appuyer sur la flèche du bas (↓) le fera reculer de 10 pas. Les flèches de droite (→) et de gauche (←) le tourneront de 10° respectivement vers la droite et vers la gauche. Donc, pour faire tourner le lutin de 90° comme sur la figure 2-9, vous devrez appuyer neuf fois sur la flèche de droite ou de gauche.

Tout d'abord, créez un nouveau projet Scratch. Remplacez le costume du chat par quelque chose qui indique clairement si le lutin regarde à droite, à gauche, en haut ou en bas. Les costumes *beetle* ou *cat2* (du dossier *Animaux*) sont de bons choix, mais vous pouvez en préférer un autre si vous le souhaitez. Dans l'onglet **Costumes**, cliquez sur le bouton **choisir un costume dans la bibliothèque** et choisissez.

Copyright © 2016 Eyrolles.

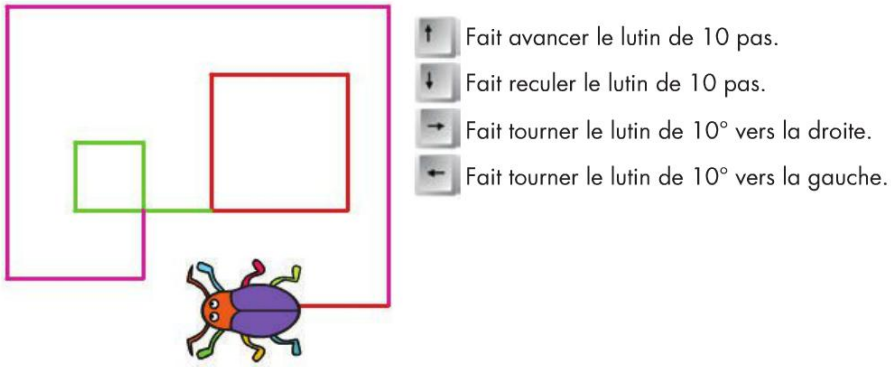


Figure 2-9 : L'application Dessin facile en action

À présent, ajoutez les scripts de la figure 2-10 à votre lutin. Vous pouvez créer les quatre blocs concernant les flèches à partir du bloc **quand espace est cliqué** de la palette *Événements*. Choisissez la flèche directionnelle dont vous avez besoin à partir du menu déroulant.

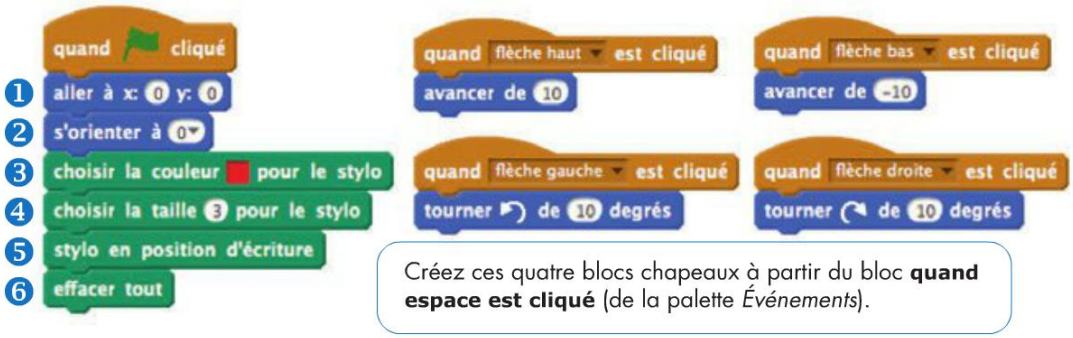


Figure 2-10 : Les scripts de l'application Dessin facile

Lorsque vous cliquez sur le drapeau vert, le lutin va au centre de la scène ❶ et s'oriente vers le haut ❷. Ensuite, la couleur du stylo ❸ ainsi que sa taille ❹ sont définies et le script place le stylo en position d'écriture ❺. Ceci fait, le programme efface tout dessin préalablement tracé dans la scène ❻.

Vous n'aurez qu'à cliquer sur le drapeau vert pour effacer la scène et commencer un nouveau dessin. Utilisez les flèches du clavier pour dessiner la forme qui vous plaira. À votre avis, que créerait la séquence →↑→↑→↑→↑… ?

FAITES LE TEST 2-4

Ajoutez une option pour faire en sorte que le stylo trace un trait plus épais lorsque la touche *E* est enfoncée et plus fin lorsque la touche *F* est enfoncée. Imaginez d'autres manières d'améliorer cette application et implémentez-les.

Copyright © 2016 Eyrolles.

Le pouvoir des répétitions

Nos programmes sont pour le moment restés assez simples, mais lorsque vous commencerez à écrire des scripts plus longs, il vous arrivera souvent de devoir recopier une même pile de blocs plusieurs fois. La duplication de scripts rend les programmes plus longs, plus difficiles à comprendre et plus délicats à expérimenter. Si vous devez changer un nombre, par exemple, vous devrez vous assurer de le modifier dans chaque copie de la pile de blocs. La commande **répéter** de la palette *Contrôle* évite ce problème.

DessinCarré.sb2

Imaginons que vous vouliez dessiner le carré présenté à la figure 2-11 (à gauche). Vous pourriez dire au lutin de suivre ces instructions répétitives :

1. Avance d'une certaine distance et tourne de 90° dans le sens des aiguilles d'une montre.
2. Avance de la même distance et tourne de 90° dans le sens des aiguilles d'une montre.
3. Avance de la même distance et tourne de 90° dans le sens des aiguilles d'une montre.
4. Avance de la même distance et tourne de 90° dans le sens des aiguilles d'une montre.

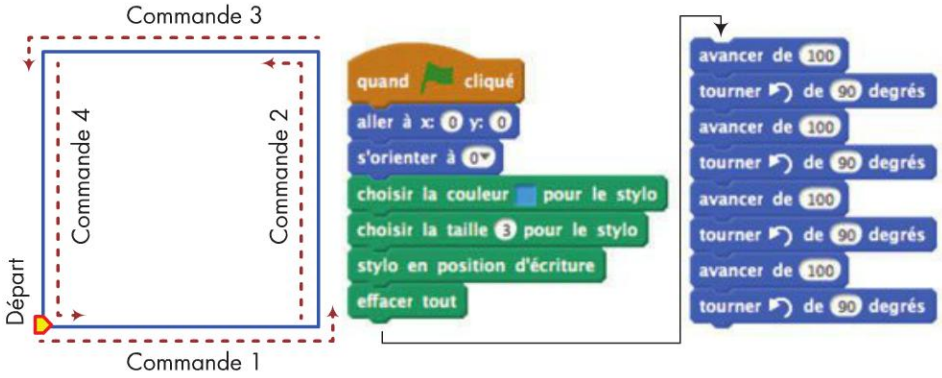


Figure 2-11 : Un carré (à gauche) et un script qui le dessine (à droite) en utilisant une suite de commandes **avancer de** *et* **tourner**

La figure 2-11 montre également un script qui implémente ces instructions. Les commandes **avancer de 100** et **tourner de 90°** sont répétées quatre fois. Nous allons éviter d'utiliser sans arrêt les mêmes blocs grâce au bloc **répéter**, lequel exécute les commandes qu'il contient autant de fois que vous lui direz de le faire (figure 2-12). Utiliser un bloc **répéter** rend également ces instructions bien plus faciles à comprendre.

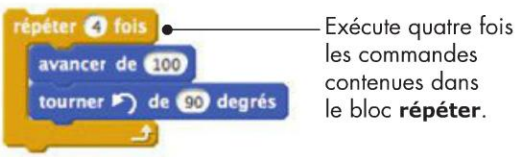


Figure 2-12 : Utilisation du bloc **répéter** *pour dessiner un carré*

Le carré dessiné avec le script de la figure 2-11 dépend de la direction que prend le lutin au démarrage. Ce concept est illustré sur la figure 2-13. Après avoir dessiné le carré, le lutin retournera à sa position initiale et se réorientera dans la direction qu'il visait avant de se déplacer.

Copyright © 2016 Eyrolles.

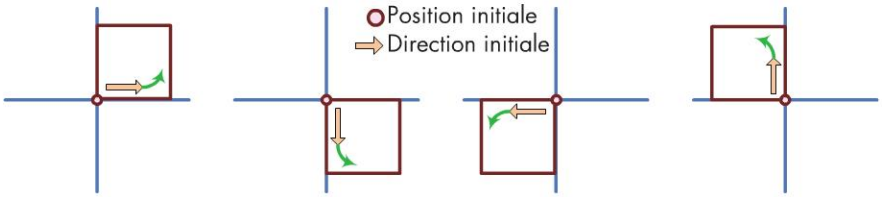
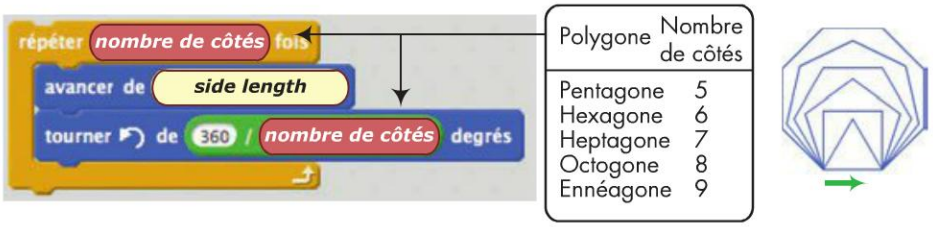


Figure 2-13 : La direction initiale du lutin change la position du carré.

Polygone.sb2

FAITES LE TEST 2-5

Vous pouvez facilement modifier le script de la figure 2-12 pour dessiner d'autres polygones classiques. Le script modifié aura la forme présentée ci-après. Vous pouvez remplacer « nombre de côtés » par n'importe quel nombre entier pour indiquer le polygone désiré. De même, remplacer « longueur des côtés » vous permettra de choisir la taille du polygone. La figure montre également six polygones dont les côtés ont la même longueur, qui ont été dessinés grâce à ce script. La position et la direction initiales sont indiquées par la flèche verte. Ouvrez le fichier *Polygone.sb2* et exécutez-le avec différentes valeurs pour « nombre de côtés ». Que se passe-t-il quand ce nombre devient très grand ? Cela devrait vous donner une idée de la façon de dessiner un cercle.



Polygone	Nombre de côtés
Pentagone	5
Hexagone	6
Heptagone	7
Octogone	8
Ennéagone	9

Carrés rotatifs

CarrésRotatifs.sb2

Vous pouvez créer de superbes dessins en répétant un motif selon une certaine séquence. Par exemple, le script de la figure 2-14 dessine un carré et le fait tourner 12 fois. Les blocs d'initialisation du stylo et de la position d'écriture ne sont pas représentés ici.

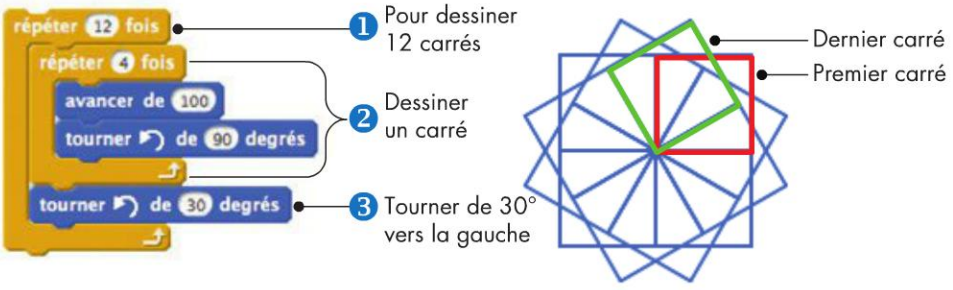


Figure 2-14 : Dessin d'un carré rotatif

Le bloc **répéter** extérieur ❶ est exécuté 12 fois. À chaque passage dans cette boucle, le lutin dessine un carré ❷ et tourne de 30° vers la gauche ❸ pour se préparer à dessiner le prochain.

Copyright © 2016 Eyrolles.

FAITES LE TEST 2-6

Vous remarquerez que (12 répétitions) × (30° pour chaque répétition) = 360°. À votre avis, que se passerait-il si vous changiez les nombres du programme à 4 répétitions et 90° ? Et pour 5 et 72° ? Observez les résultats avec différentes valeurs pour le nombre de **répéter** et l'angle de **tourner**.

L'estampillage

Moulin.sb2

Dans la section précédente, vous avez appris à utiliser les blocs **tourner** et **répéter** pour transformer de simples formes en motifs complexes. Et si vous vouliez faire tourner des formes plus compliquées ? Au lieu de dessiner la forme de base avec les commandes **avancer** et **tourner**, vous pouvez créer un nouveau costume dans l'éditeur d'images et utiliser le bloc **estampiller** pour en dessiner plusieurs copies sur la scène. Pour illustrer cette technique, écrivons le programme de la figure 2-15, avec lequel nous dessinerons un moulin à vent.



Figure 2-15 : La commande **estampiller** *facilite la création de motifs géométriques complexes.*

Nous avons dessiné le drapeau en utilisant l'éditeur d'images (figure 2-15, à gauche) et l'avons utilisé comme costume pour notre lutin. Nous en avons placé le centre en bas du mât, afin de faire tourner le drapeau autour de ce point.

Le script dessinant le moulin est représenté au milieu de la figure 2-15. Le bloc **répéter** s'exécute huit fois ; à chaque fois, il estampille une copie du costume sur la scène avant de tourner le lutin de 45° vers la gauche. Notez que, pour faire fonctionner ce script, vous devez utiliser le bloc **fixer le sens de rotation** réglé sur *360°* pour permettre au drapeau de suivre le mouvement.

NOTE *Le fichier* DrawingGeometricShapes.pdf *(que vous pourrez télécharger à l'adresse* http://www.editions-eyrolles.com/go/grandlivrescratch*) fournit des explications complètes (en anglais) sur la réalisation de dessins de formes géométriques telles que les rectangles, parallélogrammes, losanges, trapèzes, cerfs-volants et polygones, et enseigne les techniques de dessin à base de polygone.*

FAITES LE TEST 2-7

Le bloc **ajouter à l'effet** (de la palette *Apparence*) vous permet d'appliquer des effets graphiques tels que **couleur**, **tournoyer** ou **œil de poisson**. Ouvrez le fichier *Moulin.sb2* et ajoutez-y les commandes à l'intérieur du bloc **répéter**. Expérimentez d'autres effets graphiques pour créer de nouveaux motifs. Notez bien que, pour que le bloc **ajouter à l'effet couleur** fonctionne, la couleur du drapeau dans l'éditeur d'images ne doit pas être noire.

Copyright © 2016 Eyrolles.

Projets Scratch

Jump.sb2

Dans cette section, nous allons développer deux programmes courts qui devraient améliorer votre compréhension des blocs des palettes *Mouvement* et *Stylo*. Vous trouverez les lutins et arrière-plans dans les fichiers de projets. Nous nous concentrerons donc plutôt sur l'écriture des scripts qui feront tourner ces applications. L'explication (en anglais) d'un jeu bonus additionnel, appelé *Jump*, figure dans le fichier *ApplicationsBonus.pdf* (*http://www.editions-eyrolles.com/go/grandlivrescratch*).

Certains de ces scripts utiliseront des blocs de commande que vous n'avez pas encore vus, mais ne vous inquiétez pas si certaines choses vous échappent : vous les étudierez dans les chapitres suivants.

Prenez l'argent

Argent_PasDeCode.sb2

Notre première application est un jeu simple dans lequel le joueur devra déplacer le lutin avec les flèches du clavier et collecter autant de sacs d'or que possible. Comme le montre la figure 2-16, le sac apparaît à une position aléatoire sur la grille. Si le joueur ne l'attrape pas dans les 3 secondes, il se déplace à un autre endroit.

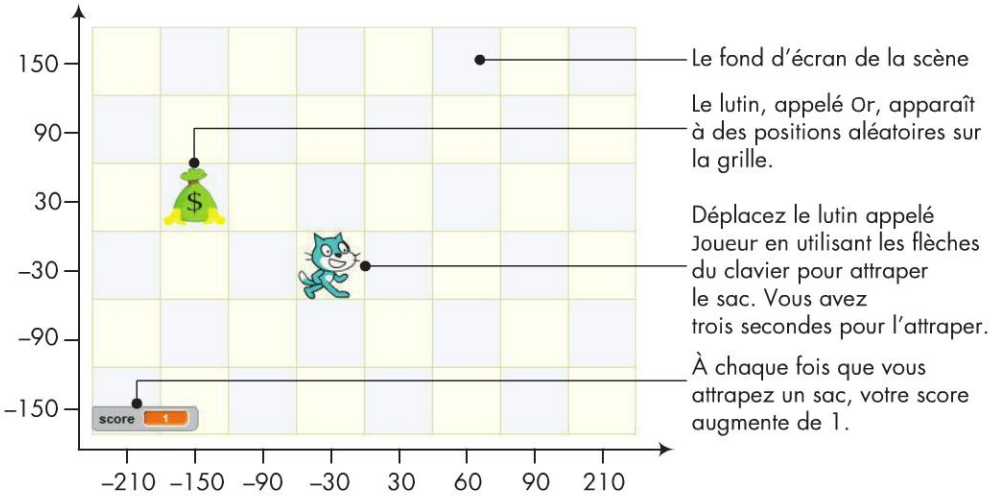


Figure 2-16 : Aidez le chat à attraper autant de sacs d'or que possible !

Ouvrez le fichier *Argent_PasDeCode.sb2*. Les scripts en sont absents, mais il contient tous les autres éléments dont vous aurez besoin.

NOTE *Les axes de coordonnées de la figure 2-16 ont été ajoutés afin de vous aider à comprendre les nombres utilisés dans ces scripts. Référez-vous à cette figure si vous ressentez le besoin de rafraîchir votre vision mentale de la façon dont se déplacent les lutins.*

Tout d'abord, écrivons les scripts du lutin Joueur (figure 2-17).

Copyright © 2016 Eyrolles.

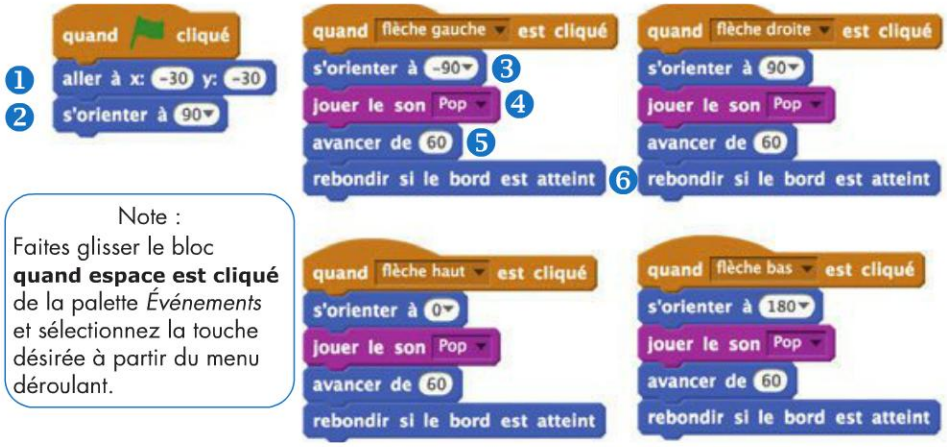


Figure 2-17 : Le script du lutin Joueur

Quand le joueur clique sur le drapeau vert, le lutin se déplace en *(-30, -30)* ❶ et s'oriente vers la droite ❷. Les quatre autres scripts répondent aux flèches du clavier. Quand une touche flèche est enfoncée, le script correspondant change la direction du lutin ❸, joue un son court (en utilisant le bloc **jouer le son** ❹ de la palette *Sons*) et fait avancer le lutin de 60 pas ❺. Si besoin, le lutin rebondit sur les bords de la scène ❻. Puisque 60 pas correspondent à un carré de la grille de la figure 2-16, à chaque fois que vous appuyez sur une flèche, le lutin Joueur se déplace d'un carré.

NOTE *Avez-vous remarqué que les 4 scripts gérant les flèches du clavier sont quasiment identiques ? Dans le chapitre 4, vous apprendrez à éviter de telles duplications de code.*

À présent, testez cette partie du jeu. Vous devriez pouvoir déplacer le lutin Joueur dans la scène grâce aux flèches directionnelles du clavier. Passons au lutin Or, dont le script est présenté à la figure 2-18.

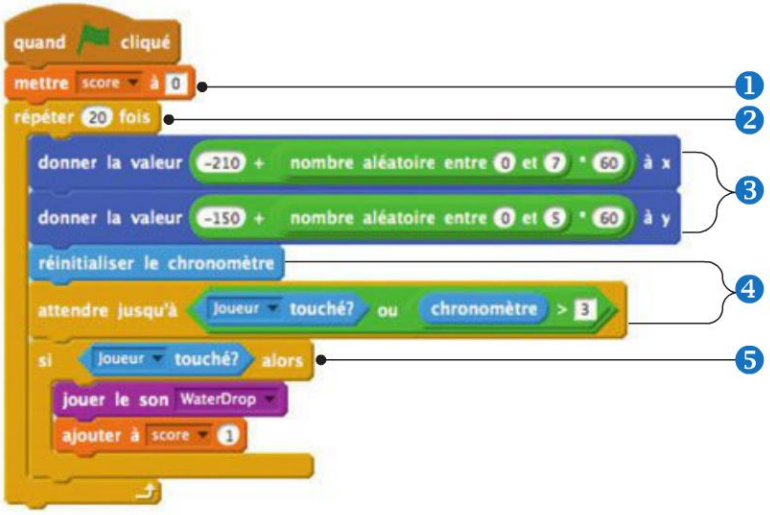


Figure 2-18 : Le script du lutin Or

Comme celui de Joueur, ce script démarre lorsqu'on clique sur le drapeau vert. Il déplace le sac d'or dans la scène. Il détecte également le nombre de sacs collectés grâce à une variable nommée score, que j'ai préalablement créée pour vous dans la palette *Données*.

Copyright © 2016 Eyrolles.

NOTE *Les valeurs telles que* score *sont appelées* variables. *Elles permettent d'enregistrer des informations à utiliser plus tard dans le programme. Vous apprendrez tout sur les variables dans le chapitre 5.*

Puisque le jeu vient de commencer et que nous n'avons pas encore attrapé de sac d'or, le score est initialisé à 0 ❶. Nous commençons ensuite une boucle qui se répétera 20 fois ❷, montrant au joueur 20 sacs d'or au total (n'hésitez pas à changer ce nombre comme bon vous semblera). À chaque fois que la boucle s'exécute, le sac d'or apparaît à une position aléatoire ❸, laisse du temps au joueur pour l'attraper ❹ et incrémente *score* si le joueur réussit ❺.

On souhaite que le sac apparaisse de façon aléatoire sur l'un des 48 carrés de la scène. Comme vous l'avez vu sur la figure 2-16, la position *x* du sac peut prendre les valeurs :-210, -150, -90, …, 210. Ces nombres sont espacés de 60 pas, donc chaque position *x* peut être trouvée à partir de -210 en calculant les expressions suivantes :

```
x = -210 + (0 × 60)
x = -210 + (1 × 60)
x = -210 + (2 × 60)
...
```

Un calcul similaire s'applique à la position *y*.

Nous pouvons définir la position *x* des sacs en générant un nombre aléatoire entre 0 et 7, que nous multiplions par 60, puis en ajoutant -210 au résultat. La figure 2-19 détaille les étapes de création du bloc **donner la valeur à x**. Le bloc **donner la valeur à y** est construit de manière similaire.

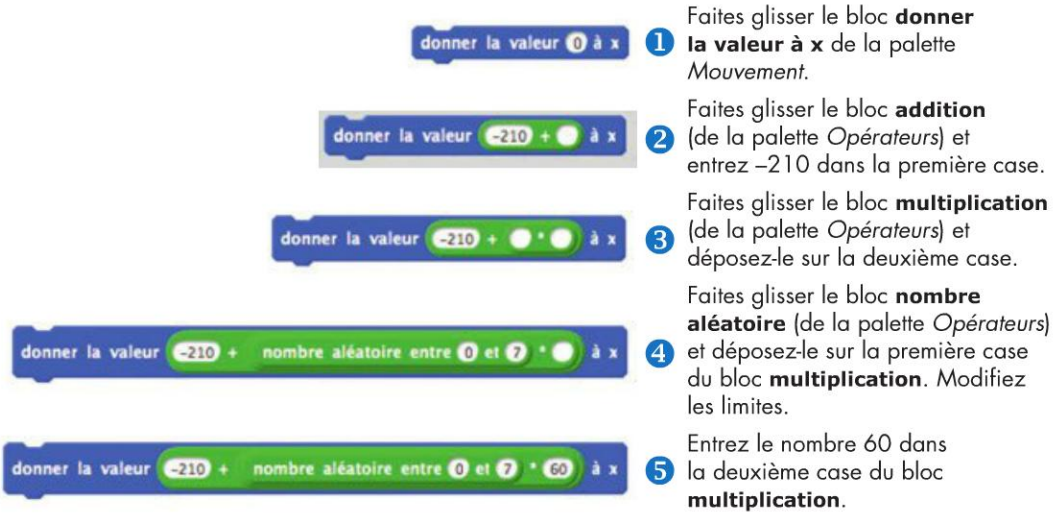


Figure 2-19 : Construction du bloc **donner la valeur à x** *de la figure 2-18*

Après être apparu à une position aléatoire, le sac d'or donne au joueur trois secondes pour l'attraper (vous pouvez modifier cette valeur pour moduler la difficulté du jeu). Pour connaître les durées, le script commence par réinitialiser le chronomètre de Scratch. Il attend ensuite que le joueur attrape le sac en le touchant ou que le chronomètre dépasse les trois secondes. Dès que l'une de ces conditions se produit, le bloc **attendre jusqu'à** laisse le

Copyright © 2016 Eyrolles.

script continuer sa route et exécuter le bloc **si/alors**. Les étapes de la création du bloc **attendre jusqu'à** sont présentées à la figure 2-20.

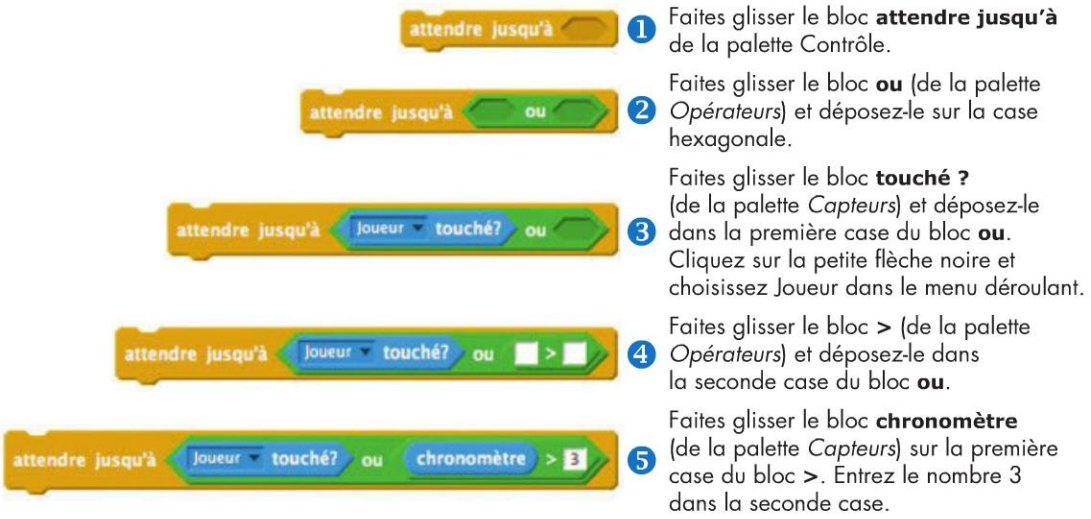


Figure 2-20 : Construction du bloc **attendre jusqu'à** *de la figure 2-18*

NOTE *Les blocs qui se trouvent à l'intérieur du* **si/alors** *ne sont exécutés que si la condition que vous spécifiez dans l'en-tête est vraie. Le chapitre 6 expliquera le fonctionnement de ce bloc plus en détail mais, pour le moment, vous en savez assez pour vous en servir et ajouter une touche personnelle à un programme.*

Si le joueur touche le sac, les commandes comprises dans le bloc **si/alors** sont exécutées. Dans ce cas, le bloc **jouer le son** émet un bruit de goutte d'eau (*WaterDrop*) et le bloc **ajouter à score** (de la palette *Données*) incrémente le score de un point.

Le jeu est à présent complet. Cliquez sur le drapeau vert pour tester votre création !

LE CHRONOMÈTRE DE SCRATCH

Un chronomètre enregistre constamment le temps écoulé depuis le démarrage de Scratch. Lorsque vous démarrez ce dernier dans un navigateur Internet, le chronomètre est initialisé à 0 et commence à compter le temps, tous les dixièmes de seconde, tant que Scratch reste ouvert. Le bloc **chronomètre** (de la palette *Capteurs*) contient le temps écoulé. La case qui l'accompagne peut être cochée/décochée pour montrer/cacher le moniteur du bloc sur la scène. Le bloc **réinitialiser le chronomètre** remet le compteur à 0 et le temps recommence à être décompté immédiatement. Le chronomètre continue à s'exécuter même si le projet ne s'exécute pas.

Attraper des pommes

AttrapePommes_PasDeCode.sb2

Considérez le jeu Attrape-pommes de la figure 2-21. Dans ce jeu, les pommes apparaissent à des positions horizontales aléatoires en haut de la scène à des

Copyright © 2016 Eyrolles.

moments aléatoires et tombent vers le sol. Le joueur doit déplacer le chariot pour les attraper avant qu'elles ne touchent le sol ; chaque pomme récoltée rapporte un point.

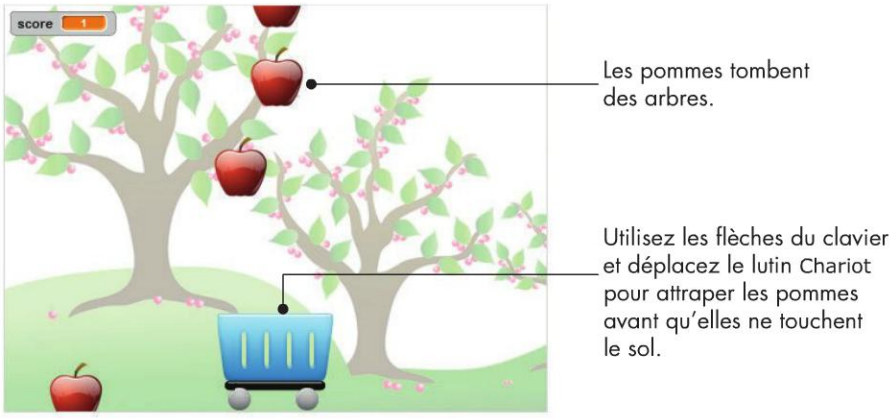


Figure 2-21 : Le jeu Attrape-pommes

A priori, vous pourriez penser qu'un tel jeu nécessite de nombreux lutins comportant des scripts quasiment identiques. En effet, il contient beaucoup de pommes. Ce n'est cependant pas le cas pour Scratch 2. Vous pouvez facilement créer un groupe de copies grâce à la fonction de *clonage*. Dans notre jeu, nous n'utiliserons qu'un seul lutin Pomme et nous créerons autant de clones que nous le désirerons. Ouvrez le fichier *AttrapePommes_PasDeCode.sb2*, qui est configuré pour démarrer la programmation de notre jeu (il ne contient pas de scripts). Pour rendre les choses un peu plus exaltantes, la configuration initiale contient également une variable nommée score (créée pour vous dans la palette *Données*), que nous utiliserons pour nous souvenir du nombre de pommes attrapées. En premier lieu, écrivez le script du lutin Chariot (figure 2-22).

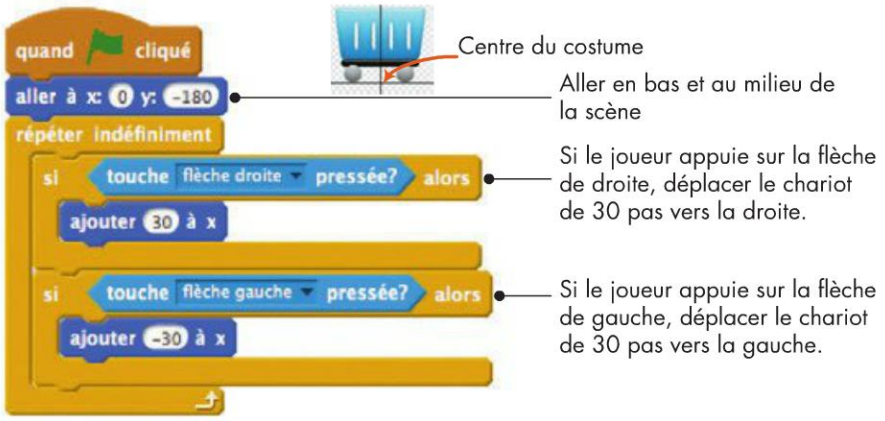


Figure 2-22 : Le script du lutin Chariot

Lors d'un clic sur le drapeau vert, nous déplaçons le chariot en bas et au milieu de la scène. Ensuite, le script vérifie constamment l'état des flèches gauche/droite et se déplace en conséquence. J'ai choisi le nombre 30 par tâtonnement ; n'hésitez pas à le modifier selon vos propres expériences.

Nous en arrivons au clonage. Commencez par ajouter le script de la figure 2-23 au lutin Pomme. Ce script démarre également lorsque le joueur clique sur le drapeau vert.

Copyright © 2016 Eyrolles.

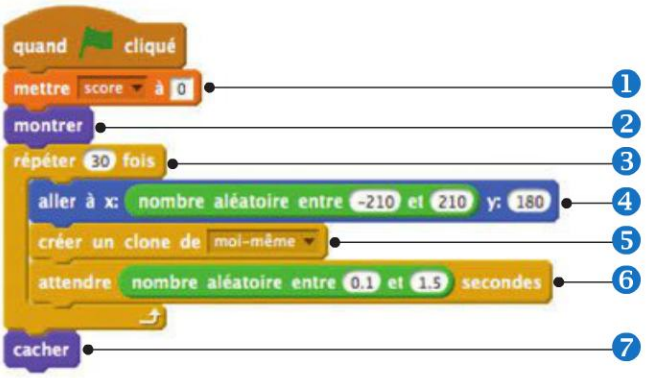


Figure 2-23 : Le premier script du lutin Pomme

Puisque nous n'avons pas encore attrapé de pomme, le script initialise la variable *score* à 0 ❶. Il rend ensuite le lutin visible grâce au bloc **montrer** de la palette *Apparence* ❷. Puis, il démarre un bloc **répéter** qui bouclera 30 fois ❸ pour faire tomber 30 pommes.

À chaque passage dans la boucle, le lutin Pomme se déplace à une position horizontale aléatoire en haut de la scène ❹. Il fait ensuite appel au bloc **créer un clone de** (de la palette *Contrôle*) pour se cloner lui-même ❺, attend pendant une courte durée aléatoire ❻ et démarre le cycle suivant du bloc **répéter**. Après avoir complété 30 cycles de ce bloc, le script masque le lutin Pomme grâce au bloc **cacher** ❼ de la palette *Apparence*.

Si vous démarrez le jeu maintenant en cliquant sur le drapeau vert, 30 pommes apparaissent aléatoirement en haut de la scène et y restent – puisque nous n'avons pas dit aux clones quoi faire. Le second script du lutin Pomme, présenté à la figure 2-24, entre alors en jeu.

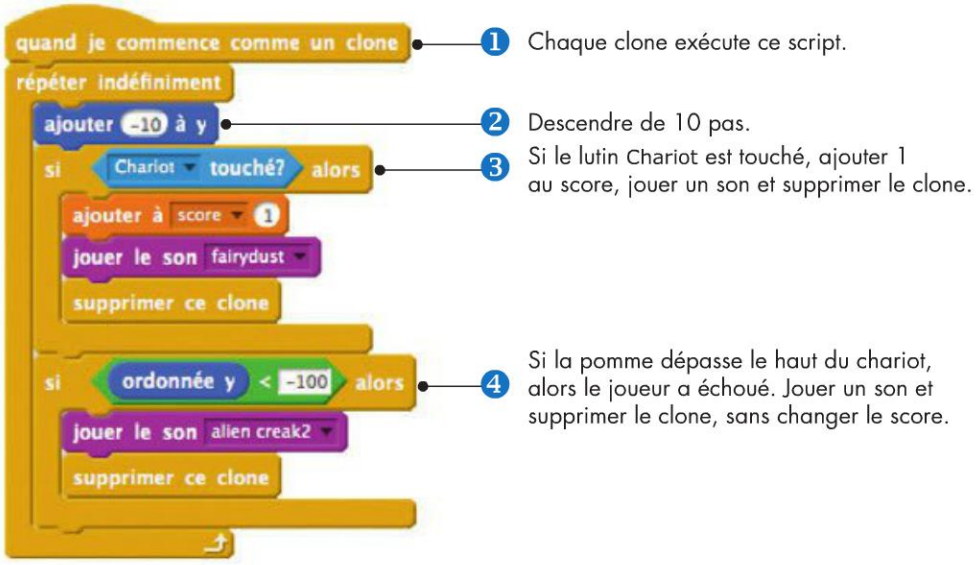


Figure 2-24 : Le second script du lutin Pomme

Grâce au bloc **quand je commence comme un clone** ❶ (de la palette *Contrôle*), chaque clone exécute le script. Chaque pomme descend de 10 pas ❷ et vérifie si elle a été attrapée ou ratée par le chariot. Si le clone détecte qu'il touche le chariot ❸, cela signifie qu'il est attrapé. Ainsi, le score

Copyright © 2016 Eyrolles.

est augmenté, un son est joué et le clone se supprime, car il n'a plus rien à faire. Si le clone tombe sous le chariot ❹, c'est que le joueur l'a raté ; dans ce cas, il joue un son différent avant de se supprimer. Si le clone n'est ni attrapé, ni raté, alors il continue à tomber et le bloc **répéter indéfiniment** continue de tourner.

Maintenant que nos pommes savent comment tomber, notre jeu est complet ! Vous pouvez le tester en cliquant sur le drapeau vert. Si vous voulez expérimenter, essayez de changer le temps d'attente entre chaque clonage de pomme et la vitesse de déplacement du chariot. Cela vous donne-t-il des idées pour modifier la difficulté du jeu ?

Plus de détails sur les lutins clonés

Tout lutin peut se copier lui-même ou copier un autre lutin en utilisant le bloc **créer un clone de**. La scène peut également cloner des lutins avec ce bloc.

Un lutin cloné hérite de *l'état* de l'original au moment où il est dupliqué – c'est-à-dire sa position et sa direction actuelles, son costume, son état de visibilité, sa couleur et sa taille de stylo, ses effets graphiques, etc. Cette idée est illustrée sur la figure 2-25.

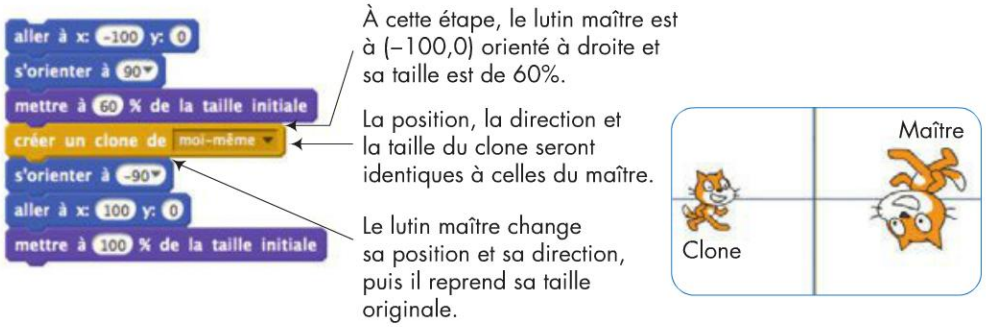


Figure 2-25 : Un clone hérite des attributs de son maître.

Les clones héritent également des scripts de leur lutin maître. Dans la figure 2-26, le lutin maître crée deux clones. Lorsque vous appuyez sur la barre d'espace, tous trois tournent de 15° vers la droite puisqu'ils exécutent tous le script **quand espace est cliqué**.

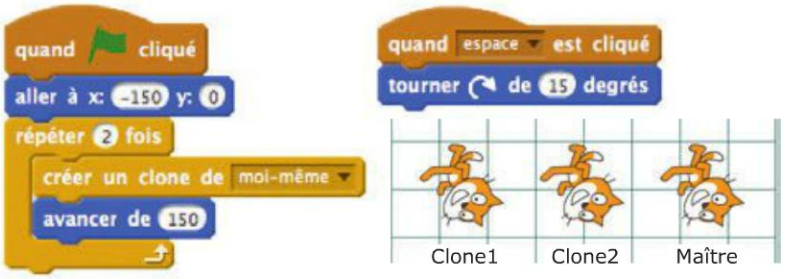


Figure 2-26 : Les clones héritent des scripts de leur maître.

Copyright © 2016 Eyrolles.

Lorsque vous utilisez le bloc **créer un clone de**, portez toujours une attention particulière aux scripts qui ne démarrent pas avec un déclenchement par drapeau vert, ou vous pourriez vous retrouver avec plus de clones que vous ne le désiriez. Considérez le programme de la figure 2-27. La première fois que vous appuyez sur la barre d'espace, un clone est créé et l'application contient deux lutins (le maître et son clone).



Figure 2-27 : Clonage à partir d'un événement d'appui de touche

À présent, si vous appuyez sur la barre d'espace une deuxième fois, vous obtenez quatre lutins dans votre application. En effet, le lutin maître mais aussi le premier clone créent chacun un clone. Si vous appuyez une troisième fois sur la barre d'espace, vous obtenez huit lutins dans votre application. Le nombre de clones grandit de façon exponentielle…

Vous pouvez résoudre ce problème en ne clonant les lutins que dans des scripts commençant par le bloc **quand le drapeau vert est cliqué**. Ces scripts ne sont exécutés qu'une fois et seulement par le lutin maître.

Résumé

Vous avez appris dans ce chapitre à déplacer vos lutins vers des points spécifiques de la scène en utilisant les commandes de mouvement absolu. Vous avez ensuite utilisé les commandes de mouvement relatif pour déplacer vos lutins par rapport à leurs position et direction actuelles. Après cela, vous avez créé de superbes dessins grâce aux commandes de stylo.

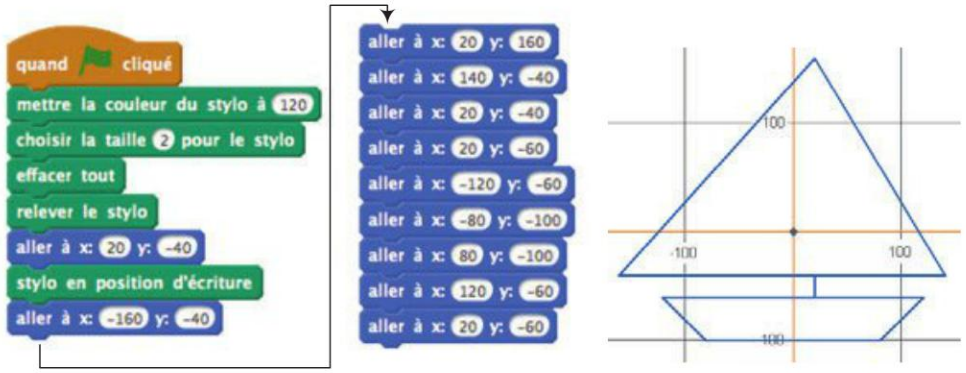
En dessinant différentes formes, vous avez découvert la puissance du bloc **répéter**, qui vous aide à créer des scripts plus courts et plus efficaces. Vous avez également pris connaissance de la commande **estampiller** et l'avez utilisée avec le bloc **répéter** pour concevoir facilement des motifs complexes.

À la fin de ce chapitre, vous avez créé deux jeux et avez fait connaissance avec la fonction de clonage. Dans le chapitre suivant, vous utiliserez les palettes *Apparence* et *Sons* pour créer des programmes encore plus attrayants.

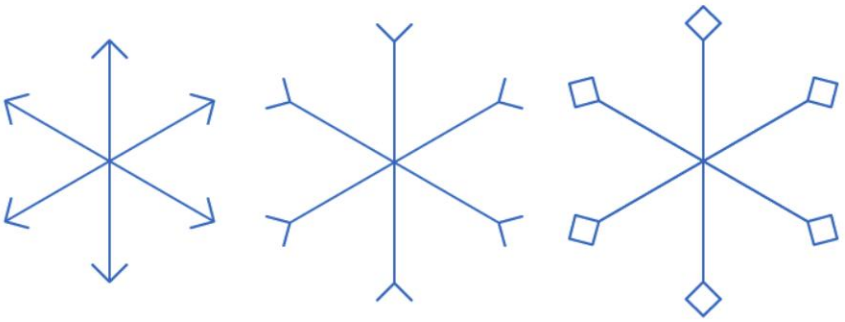
Copyright © 2016 Eyrolles.

Problèmes

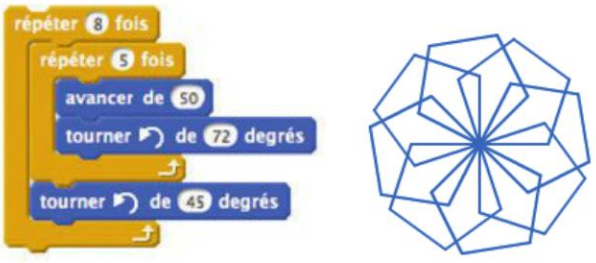
1. Expliquez le fonctionnement du script suivant. Écrivez les coordonnées *(x,y)* à chaque coin de la figure.



2. Écrivez un script qui connecte chacune des suites de points suivantes dans l'ordre afin de dévoiler la forme finale.
   a. *(30,20), (80,20), (80,30), (90,30), (90,80), (80,80), (80,90), (30,90), (30,80), (20,80), (20,30), (30,30), (30,20)*
   b. *(-10,10), (-30,10), (-30,70), (-70,70), (-70,30), (-60,30), (-60,60), (-40,60), (-40,10), (-90,10), (-90,90), (-10,90), (-10,10)*
3. Écrivez un script qui dessinera chacun des motifs suivants.

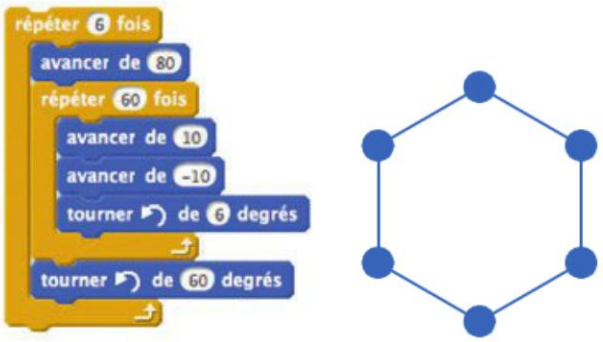


4. Considérez le script ci-après, ainsi que sa sortie. Ajoutez-y les commandes de configuration du stylo nécessaires, exécutez-le et expliquez comment il fonctionne.

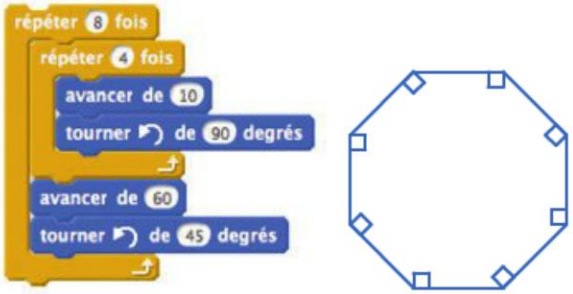


Copyright © 2016 Eyrolles.

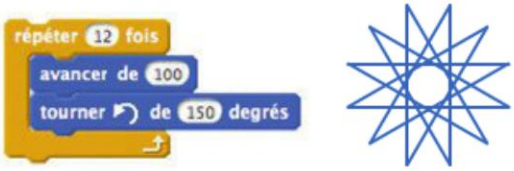
5. Considérez le script ci-après, ainsi que sa sortie. Ajoutez-y les commandes de configuration du stylo nécessaires, exécutez-le et expliquez comment il fonctionne.



6. Considérez le script ci-après, ainsi que sa sortie. Ajoutez-y les commandes de configuration du stylo nécessaires, exécutez-le et expliquez comment il fonctionne.



7. Considérez le script ci-après, ainsi que sa sortie. Ajoutez-y les commandes de configuration du stylo nécessaires, exécutez-le et expliquez comment il fonctionne.



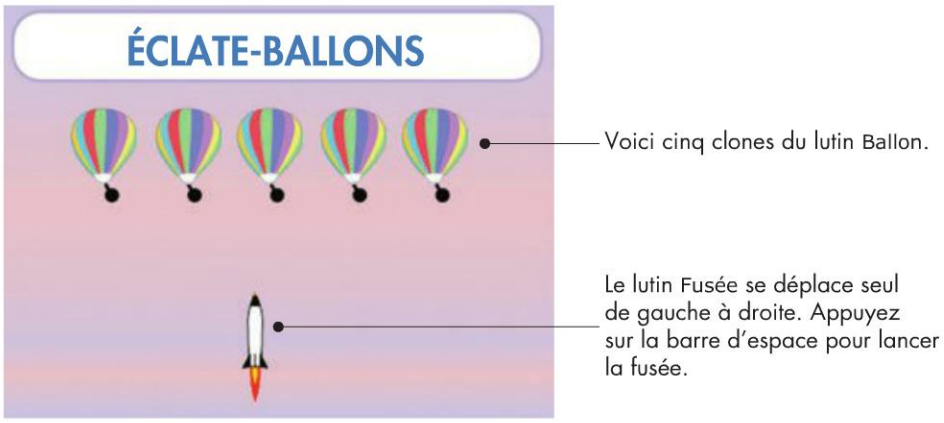
8. Écrivez un programme qui produit la sortie ci-après.



Copyright © 2016 Eyrolles.

EclateBallon_PasDeCode.sb2

9. Dans ce problème, vous écrirez le script nécessaire à la finalisation du jeu Éclate-ballons présenté ci-après.



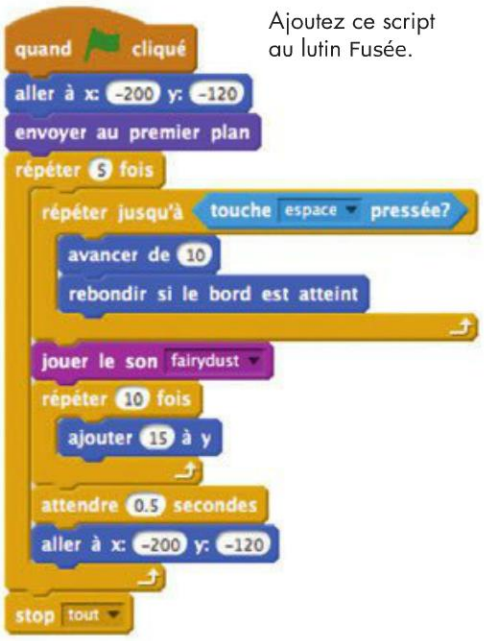
Ce jeu contient deux lutins, nommés Ballon et Fusée. Lorsque vous cliquez sur le drapeau vert, Ballon crée cinq clones dans l'interface. Le lutin Fusée se déplace seul de gauche à droite, en rebondissant sur les bords de la scène. Vous devez appuyer sur la barre d'espace au bon moment pour lancer la fusée et toucher les ballons.

Ouvrez le fichier *EclateBallon_PasDeCode.sb2*. Il contient le code permettant de créer les cinq clones lorsque le jeu démarre. Vous devez le compléter en y ajoutant les deux scripts suivants.

Ajoutez ce script au lutin Ballon.



Ajoutez ce script au lutin Fusée.



Copyright © 2016 Eyrolles.

3

APPARENCE ET SONS

Maintenant que vous savez déplacer les lutins sur la scène et utiliser le stylo pour dessiner des motifs, nous allons aborder les différentes commandes des palettes *Apparence* et *Sons*. Dans ce chapitre, vous allez :

- créer des animations et des effets graphiques ;
- apprendre comment fonctionnent les plans dans Scratch ;
- lire un fichier audio et composer de la musique ;
- créer vos propres scènes animées.

Les commandes de la palette *Apparence* vous serviront à créer des animations et à appliquer des effets graphiques aux costumes et aux arrière-plans (tournoyer, œil de poisson, fantôme). Avec les commandes de la palette *Sons*, vous ajouterez des sons, des voix ou de la musique à votre application. Plongeons au cœur du sujet avec une animation !

Copyright © 2016 Eyrolles.

La palette Apparence

Vous pouvez dessiner des images directement sur la scène en utilisant les commandes du stylo, mais les costumes fournissent un autre moyen puissant, et parfois bien plus simple, d'ajouter des éléments graphiques à vos programmes. Les commandes de la palette *Apparence* vous permettront de manipuler les costumes pour créer des animations, d'ajouter effets graphiques ou bulles de bande dessinée et de modifier la visibilité d'un lutin.

Animer en changeant de costume

Animation.sb2

Vous savez comment acheminer un lutin d'un point à un autre sur la scène. Cependant, les lutins statiques ne semblent pas très réalistes lorsqu'ils se déplacent. En basculant assez rapidement entre plusieurs costumes, vous pouvez donner l'illusion que le lutin bouge réellement ! Ouvrez le fichier *Animation.sb2* pour essayer l'animation présentée à la figure 3-1.



Figure 3-1 : Vous pouvez créer l'illusion d'une animation en basculant d'un costume à l'autre.

Cette application contient un lutin avec sept costumes et un script, visibles dans leurs onglets respectifs. Quand vous lancez l'application en cliquant sur le drapeau vert, le personnage semble marcher sur la scène. Le point clé qui permet ce mouvement est la commande **costume suivant**, qui indique au lutin de prendre le costume suivant dans sa liste et de reprendre le premier lorsqu'il atteint la fin de la liste.

Lors d'un clic sur le drapeau vert, le script démarre une boucle **répéter indéfiniment** avec un bloc **attendre** qui crée une pause de 0,1 seconde après chaque changement de costume. Si vous retirez cette pause du script, le personnage se met à courir au lieu de marcher. Essayez différentes valeurs pour les blocs **avancer de** et **attendre** ; observez la façon dont l'animation en est modifiée.

Bien sûr, vous pourriez dessiner ce personnage qui marche avec les commandes du stylo, mais cela vous demanderait d'écrire un long script. De plus, une fois les costumes dessinés, programmer une animation est un jeu d'enfant. N'oubliez pas qu'il est possible de créer des images avec votre programme d'édition d'images favori, ou encore l'éditeur d'images de Scratch.

Copyright © 2016 Eyrolles.

ClicLeVisage.sb2

Si vous souhaitez que les utilisateurs puissent interagir avec votre lutin, vous pouvez changer son costume lors d'un clic, comme dans l'application *ClicLeVisage.sb2*. Cette dernière contient un unique lutin, nommé Visage, qui possède les cinq costumes présentés à la figure 3-2. Il utilise le bloc **quand ce lutin est cliqué** (de la palette *Événements*) pour changer de costume.

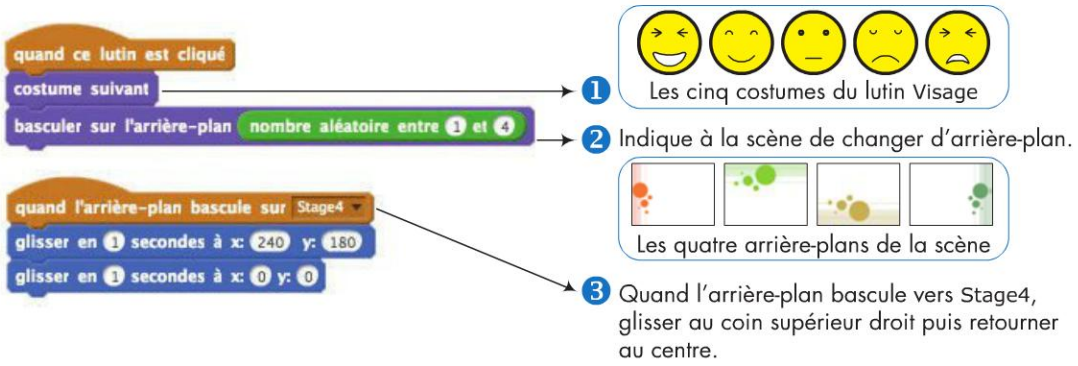


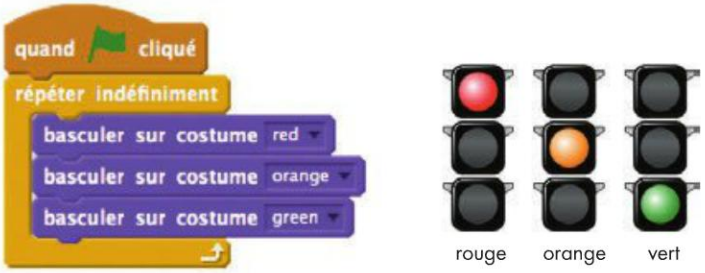
Figure 3-2 : Le visage et l'arrière-plan changent dès qu'on clique sur le lutin.

Lorsque vous lancez cette application, chaque clic sur le visage change son image par l'image suivante contenue dans la liste. Le script utilise également le bloc **basculer sur l'arrière-plan** pour indiquer à la scène de basculer aléatoirement vers l'un de ses quatre arrière-plans. Lorsque l'arrière-plan bascule vers son image Stage4, le lutin Visage détecte cet événement (grâce au bloc de déclenchement **quand l'arrière-plan bascule sur** de la palette *Événements*). Dans ce cas, le visage fait une excursion vers le coin supérieur droit, puis retourne au centre.

FAITES LE TEST 3-1

FeuRouge.sb2

Le fichier *FeuRouge.sb2* contient un lutin qui possède trois costumes (nommés *rouge*, *orange* et *vert*) et un script incomplet. Complétez l'application en y ajoutant les blocs **attendre** nécessaires à une animation de feu de circulation réaliste.



Copyright © 2016 Eyrolles.

NOTE *Utilisez la commande* **basculer sur l'arrière-plan** *pour changer de scène dans une histoire, passer au niveau suivant dans un jeu et ainsi de suite. Tous les lutins de votre projet peuvent se servir du bloc* **quand l'arrière-plan bascule sur** *pour détecter le moment où la scène a basculé sur un certain costume et agir en conséquence. Pour plus de détails, référez-vous à la fenêtre Aide de l'interface de Scratch.*

Des lutins qui pensent et qui parlent

Comme sur la figure 3-3, utilisez les blocs **dire** et **penser à** pour faire parler ou penser votre lutin, comme le ferait un personnage de bande dessinée.

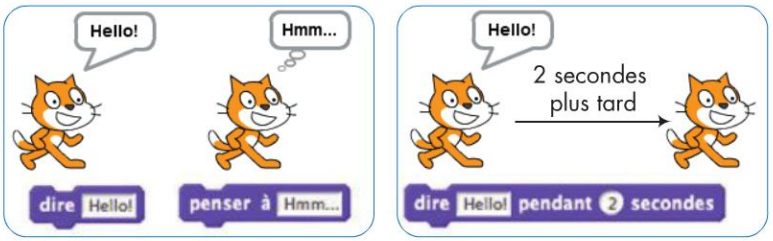


Figure 3-3 : Utilisez les blocs **dire** *et* **penser à** *pour afficher un message dans une bulle de bande dessinée.*

Les phrases que vous saisissez dans les zones de texte de ces blocs apparaissent au-dessus du lutin de façon permanente. Si vous voulez effacer le message, utilisez un bloc **dire** ou **penser à** dépourvu de texte. Vous pouvez également afficher un message pour une durée définie avec le bloc **dire pendant** ou **penser pendant** (figure 3-3, à droite).

FAITES LE TEST 3-2

Dispute.sb2

Pour voir les commandes **dire** et **penser à** en action, ouvrez le fichier *Dispute.sb2* et exécutez-le. Cette application simule une dispute sans fin entre deux personnages. Étudiez les scripts pour comprendre comment les actions des deux personnages sont synchronisées.



Effets graphiques

EffetsGraphiques.sb2

La commande **mettre l'effet à** applique différents effets graphiques aux costumes et aux arrière-plans. Scratch fournit des effets nommés œil de poisson, tournoyer, mosaïque, etc. (figure 3-4).

Copyright © 2016 Eyrolles.



Figure 3-4 : Cette figure montre ce qui arrive au chat lorsque vous lui appliquez les effets graphiques de Scratch.

Cliquez sur la petite flèche noire du bloc **mettre l'effet à** pour choisir ce que vous désirez appliquer. Vous pouvez également utiliser la commande **ajouter à l'effet** pour ajuster un effet au lieu de le régler directement. Par exemple, si l'effet fantôme est réglé à 40, lui ajouter 60 le réglera à 100, ce qui fera disparaître le lutin (comme un fantôme). Si vous voulez rendre son état original à une image, servez-vous du bloc **annuler les effet graphiques**.

NOTE *Vous pouvez appliquer plusieurs effets à un lutin en utilisant des séquences d'effets graphiques différents.*

Taille et visibilité

ChatQuiEternue.sb2

Vous aurez parfois besoin de changer la taille d'un lutin ou de contrôler le moment où il apparaît dans le programme. Par exemple, vous pouvez obtenir des objets qui semblent plus proches en les agrandissant, ou encore ne montrer un lutin « instructions » qu'au démarrage du jeu.

Pour agrandir ou rétrécir un lutin, vous disposez des commandes **mettre à % de la taille initiale** et **ajouter à la taille**. La première définit un lutin par rapport à sa taille originale, la seconde modifie sa taille actuelle par une valeur spécifique. Si vous devez faire apparaître ou disparaître un lutin, utilisez les blocs **montrer** et **cacher**.

Pour voir ces commandes en action, ouvrez le fichier *ChatQuiEternue.sb2*. Dans cette application, vous trouverez un chat qui éternue en changeant sa taille (figure 3-5).

Copyright © 2016 Eyrolles.

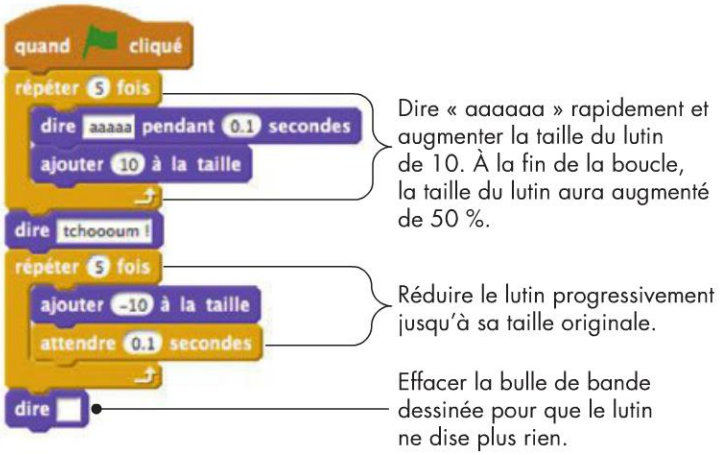


Figure 3-5 : Ce script fait éternuer le chat.

La taille du lutin augmente lorsqu'il se prépare à éternuer et retourne doucement à sa valeur originale une fois l'éternuement passé. Exécutez ce programme et observez ce qui se produit pour comprendre ces commandes.

FAITES LE TEST 3-3

Ajoutez un bloc à la fin du script de la figure 3-5 pour que le lutin Chat finisse son éternuement en disparaissant. Ajoutez un autre bloc pour faire apparaître le lutin au démarrage du script.

Plans

Les deux dernières commandes de la palette *Apparence* modifient l'ordre dans lequel les lutins sont affichés sur la scène. Cet ordre détermine quels lutins seront visibles lorsqu'ils se chevaucheront. Par exemple, imaginons que vous vouliez créer une scène dans laquelle une fille se tient derrière un rocher. Il y a deux possibilités de superposition, présentées à la figure 3-6 (à gauche).

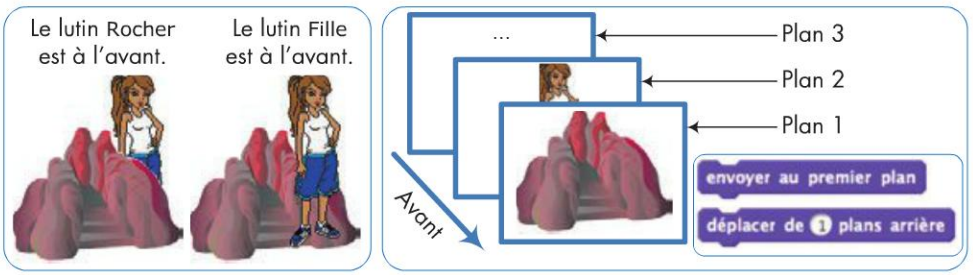


Figure 3-6 : Le lutin qui se trouve à l'avant est complètement visible et peut cacher certaines parties des lutins qu'il chevauche.

Copyright © 2016 Eyrolles.

Si vous voulez que la fille soit derrière le rocher, vous devez envoyer ce dernier au plan avant ou envoyer la fille au plan arrière. Scratch fournit deux commandes pour cela : **envoyer au premier plan** et **déplacer de plans arrière** (figure 3-6). La première indique à Scratch de mettre un lutin au-dessus de tous les autres (au premier plan) ; la seconde envoie le lutin un certain nombre de plans en arrière.

Plans.sb2

FAITES LE TEST 3-4

L'application *Plans.sb2* contient quatre objets qui se déplacent sur la scène. Vous pouvez envoyer un objet au-dessus des autres en appuyant sur la première lettre de sa couleur. Exécutez l'application pour comprendre l'effet de la commande **envoyer au premier plan**.

Nous avons vu comment créer des animations avec la palette *Apparence*, mais il y a encore une chose qui rendra vos applications plus vivantes. Dans la section suivante, nous explorerons la palette *Sons* et son ensemble de commandes.

La palette Sons

Ajouter des effets sonores ou de la musique de fond rend les jeux ou d'autres applications plus exaltants. Dans cette partie, vous apprendrez à utiliser les blocs sonores de Scratch, en commençant par incorporer des fichiers audio et contrôler leur lecture. Vous verrez ensuite des blocs de commande permettant de jouer des percussions et d'autres instruments de musique. Ensuite, vous découvrirez comment contrôler le volume et changer la vitesse (ou le tempo) à laquelle les notes de musique et les percussions seront jouées.

Jouer un fichier audio

Vous pouvez sauvegarder des fichiers audio sur votre ordinateur dans beaucoup de formats, mais Scratch n'en reconnaît que deux : WAV et MP3. Il y a trois blocs de commande qui utilisent ces fichiers audio dans votre application : **jouer le son**, **jouer le son jusqu'au bout** et **arrêter tous les sons**. **Jouer le son** laisse la commande qui la suit s'exécuter avant que le son ait fini d'être joué, alors que **jouer le son jusqu'au bout** bloque l'exécution du script jusqu'à ce que la lecture du son soit terminée. La commande **arrêter tous les sons** éteint immédiatement tous les sons en cours de lecture.

Vous pouvez ajouter une musique de fond à votre application en jouant un fichier audio de façon répétée. Pour ce faire, le plus simple est d'utiliser **jouer le son jusqu'au bout** et de recommencer (figure 3-7, à gauche).

Copyright © 2016 Eyrolles.

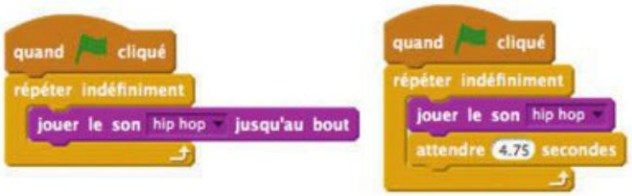


Figure 3-7 : Deux façons de jouer une musique de fond : répéter le son une fois qu'il est fini (à gauche) ou redémarrer le son après un certain laps de temps (à droite).

Selon le fichier audio, cette approche peut engendrer une pause, très courte mais remarquable, entre la fin et le redémarrage d'un son. En utilisant **jouer le son** combinée à une commande **attendre**, vous aurez un meilleur contrôle de la durée de lecture (figure 3-7, à droite). En testant plusieurs valeurs, vous rétrécirez le temps de pause et produirez une transition plus douce entre la fin de la lecture actuelle et le début de la suivante.

Jouer des percussions et d'autres sons

BeatsDemo.sb2

Si vous développez des jeux, vous voudrez probablement ajouter de courts effets sonores lorsque le joueur atteint une cible, finit un niveau, ou autre. Il est facile de créer ces sons grâce à la commande **jouer du tambour pendant x temps**, qui jouera votre choix parmi 18 sons de percussion, à un rythme correspondant à un certain nombre de temps (un temps correspond à un battement rythmique). Vous pouvez aussi ajouter des silences avec la commande **faire une pause pour x temps**. L'application *BeatsDemo.sb2* (figure 3-8) montre l'effet du paramètre de temps.

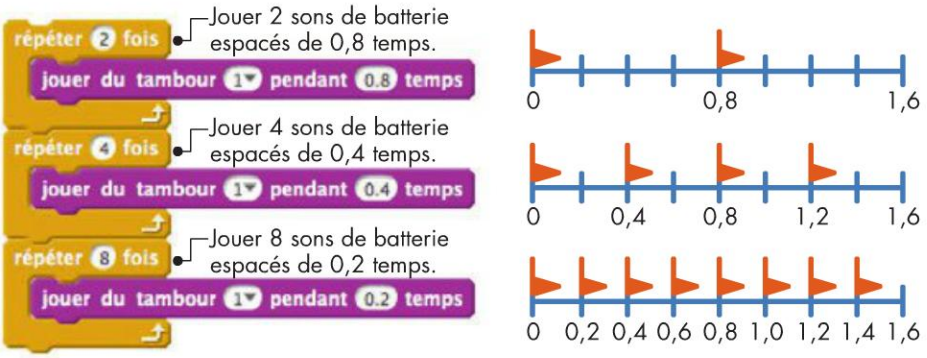


Figure 3-8 : Une illustration des temps dans Scratch

Le script contient trois blocs **répéter** avec deux, quatre et huit occurrences. Chaque bloc joue le même son de batterie, mais à un rythme (nombre de temps) différent. Imaginez l'axe temporel divisé en intervalles de 0,2 unités (figure 3-8, à droite). La première boucle joue deux sons de batterie espacés de 0,8 unités. La deuxième boucle joue quatre sons de batterie espacés de 0,4 unités. La troisième joue huit sons de batterie espacés de 0,2 unités. Chaque boucle a la même durée d'exécution ; nous frappons simplement les percussions plus ou moins rapidement dans un même laps de temps.

Copyright © 2016 Eyrolles.

J'ai utilisé le terme « unités temporelles » au lieu de secondes car le temps nécessaire à l'exécution d'une boucle dépend du *tempo*, que vous définissez avec la commande **mettre le tempo à**. Si vous utilisez le tempo par défaut de 60 battements par minute (bpm), chaque boucle de la figure 3-8 s'exécute en 1,6 seconde. Si vous réglez le tempo à 120 bpm, chaque boucle dure 0,8 seconde. Le régler à 30 bpm porte la durée à 3,2 secondes.

Composer de la musique

FrèreJacques.sb2

Scratch contient également deux commandes qui jouent des notes de musique pour composer ainsi votre propre musique. La commande **jouer la note pendant x temps** fait entendre la note de votre choix, comprise entre 0 et 127, pendant le nombre de temps indiqué. La commande **choisir l'instrument n°** indique à Scratch quel timbre choisir. Créons une chanson complète. Le script de la figure 3-9 joue la célèbre comptine « Frère Jacques ».

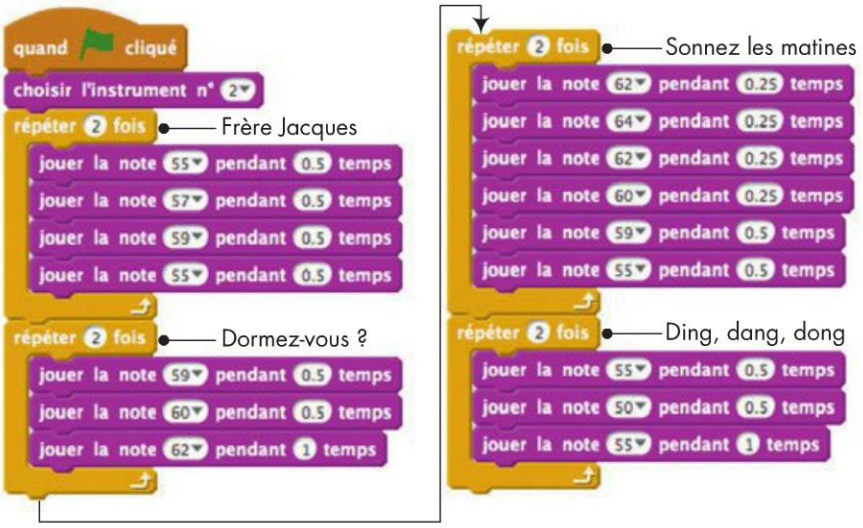


Figure 3-9 : Un script qui joue « Frère Jacques »

Ouvrez l'application *FrèreJacques.sb2* et testez-la avec différentes valeurs pour le bloc **choisir l'instrument n°** qui permet de changer le timbre du morceau.

Contrôle du volume sonore

Imaginons que vous vouliez qu'un son s'estompe en réponse à un événement de votre application. Si vous lancez une fusée dans l'espace, par exemple, vous pouvez désirer que le son de ses moteurs soit fort au décollage et devienne de plus en plus silencieux au fur et à mesure que la fusée s'éloigne.

Scratch contient un ensemble de commandes pour contrôler le volume d'un fichier audio, d'un son percussif ou d'une note de musique. La commande **mettre le volume au niveau %** définit le volume d'un lutin comme un pourcentage de celui des haut-parleurs. Cependant, elle n'affecte que ce lutin (ou la scène) ; donc, si vous voulez que des sons soient joués en même temps et à des volumes différents, vous devrez utiliser plusieurs lutins. Le bloc **ajouter au volume** augmente (nombre positif) ou baisse (nombre

Copyright © 2016 Eyrolles.

négatif) le volume de la valeur que vous précisez. Pour afficher le volume d'un lutin, cochez la case qui accompagne le bloc **volume**. Ces blocs sont bien pratiques si vous souhaitez changer le volume d'un lutin par rapport à la distance le séparant de la cible qu'il doit atteindre (comme dans un jeu de chasse au trésor) ou encore augmenter le volume de certaines parties d'une chanson. Vous pouvez aussi utiliser ces blocs pour simuler un orchestre en jouant plusieurs instruments (avec des volumes différents) simultanément.

VolumeDemo.sb2

FAITES LE TEST 3-5

Le fichier *VolumeDemo.sb2* simule un chat qui marche dans une forêt. Cette application se sert de la commande **ajouter au volume** pour diminuer progressivement le volume du chat au fur et à mesure qu'il s'enfonce dans la forêt. Trouvez des solutions pour rendre cette simulation encore plus réaliste et essayez de les implémenter.

Régler le tempo

Les trois derniers blocs de la palette *Sons* sont liés au tempo (vitesse) auquel les percussions et les notes sont jouées. Le tempo est mesuré en battements par minute (bpm). Dans Scratch, un battement est appelé *un temps*. Plus le tempo est élevé, plus les notes et les percussions seront jouées vite.

Scratch vous laisse choisir un tempo spécifique avec la commande **mettre le tempo à bpm**. Vous pouvez aussi dire à un lutin de l'augmenter ou de le diminuer d'une certaine valeur, avec la commande **ajouter au tempo**. Cochez la case qui accompagne le bloc **tempo** pour en afficher la valeur sur la scène.

TempoDemo.sb2

FAITES LE TEST 3-6

Ouvrez le fichier *TempoDemo.sb2* et exécutez-le pour voir les commandes **mettre le tempo à bpm** et **ajouter au tempo** en action.

Projets Scratch

Les commandes des palettes *Apparence* et *Sons* ajoutent de plaisants effets à vos applications. Dans cette section, nous allons rassembler tout ce que nous avons appris dans ce chapitre pour créer les scènes animées d'une personne qui danse et d'un feu d'artifice. Ceci devrait vous aider à réviser quelques nouveaux blocs de commande et à parfaire votre entraînement en créant des projets Scratch complets.

Copyright © 2016 Eyrolles.

Danser sur scène

DanseSurScene.sb2

Dans cette section, vous allez animer un lutin Danseur sur la scène. Cette application est illustrée par la figure 3-10 et le script complet est enregistré dans le fichier *DanseSurScene.sb2*. Nous allons à présent construire la scène complète.

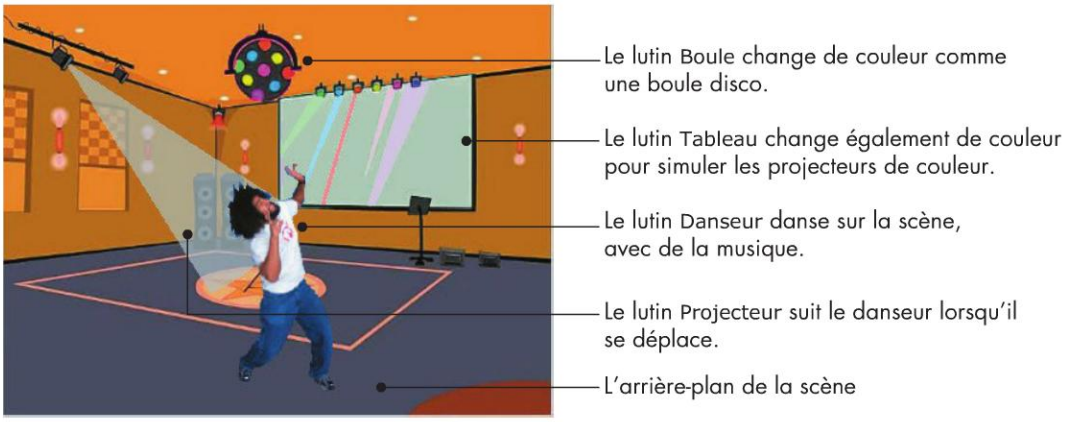


Figure 3-10 : L'application Danse sur scène en action

En premier lieu, commencez un nouveau projet. Si Scratch n'est pas encore ouvert, tout ce que vous avez à faire est de le démarrer – cela crée un nouveau projet automatiquement. Sinon, sélectionnez **Nouveau** dans le menu **Fichier**. Dans tous les cas, vous disposerez d'un nouveau projet qui contient par défaut le lutin Chat.

L'arrière-plan à utiliser est nommé *party room* dans la catégorie *intérieur*. Importez-le et supprimez l'arrière-plan blanc initial, dont vous n'aurez pas besoin. La scène devrait ressembler à la figure 3-11.

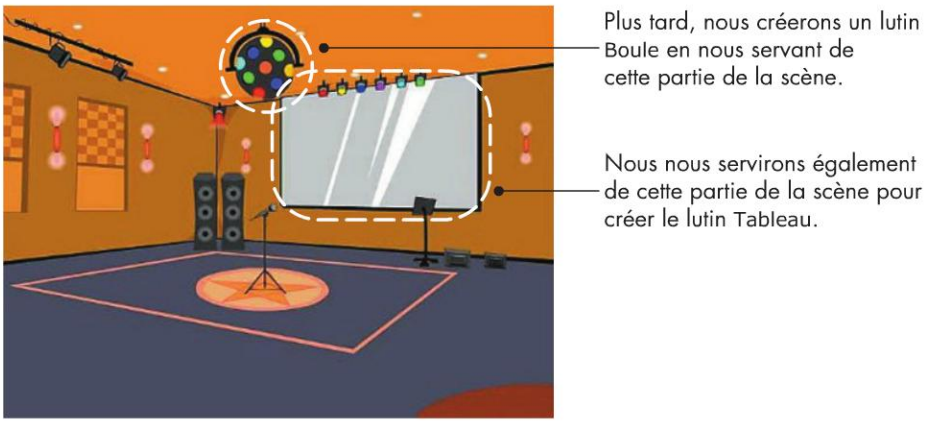


Figure 3-11 : Nous transformerons certaines parties de l'arrière-plan de la scène en lutins un peu plus tard.

En examinant les figures 3-10 et 3-11 avec attention, vous remarquerez que les lutins Boule et Tableau semblent faire partie de l'arrière-plan de la scène. Comme vous le verrez bientôt, ces deux lutins ont effectivement été créés à partir de cette image et placés sur la scène pour couvrir les parties d'où ils venaient. Créer des lutins de cette façon permet de changer leurs couleurs et de rendre la scène plus réaliste.

Copyright © 2016 Eyrolles.

Nous avons à présent besoin d'une musique de fond. Utilisons le fichier *medieval1* de la catégorie *Boucles musicales*. Importez ce fichier dans la scène et supprimez le son *pop* par défaut. Ajoutez-y ensuite le script de la figure 3-12. Il utilise la commande **jouer le son** ainsi qu'un temps d'attente redémarrant le fichier audio en douceur ; ce temps de 9,5 s a été sélectionné par expérimentation.



Figure 3-12 : La scène joue notre musique de fond.

Cliquez sur le drapeau vert pour tester ce que vous avez créé. Vous devriez entendre un clip audio se répéter continuellement. Arrêtez le script lorsque vous êtes prêt ; nous allons ajouter notre danseur.

Remplacez les costumes du lutin Chat par ceux du Danseur. Importez dan-a et dan-b de la catégorie *Gens*, supprimez les deux costumes Chat et modifiez le nom du lutin en Danseur. Le script du Danseur est représenté sur la figure 3-13.

Le danseur se déplace de 20 pas vers la droite, change de costume, se déplace de 20 pas vers la gauche et change à nouveau de costume. Ces étapes sont répétées indéfiniment pour faire en sorte qu'il ait vraiment l'air de danser. Le script change légèrement l'effet œil de poisson à chaque pas pour faire varier les choses. Cliquez sur le drapeau vert pour tester ce nouvel ajout au programme. Vous devriez entendre la musique de fond et voir le danseur se déplacer à gauche et à droite sur la scène.



Figure 3-13 : Ce script indique au lutin comment danser.

Maintenant que nous disposons d'un danseur pour notre fête, ajoutons des lumières de couleur avec les lutins Boule, Tableau et Projecteur. Pour créer le lutin Boule, cliquez sur la miniature de la scène, puis sélectionnez l'onglet **Arrière-plans**. Cliquez droit sur la miniature de l'arrière-plan *party room* et sélectionnez **enregistrer localement comme fichier** à partir du menu contextuel. Une fenêtre s'ouvre, dans laquelle vous pouvez sauvegarder l'image de l'arrière-plan à l'emplacement de votre choix. Souvenez-vous de l'endroit où vous l'avez sauvegardée, car vous la réimporterez dans un instant.

Copyright © 2016 Eyrolles.

Cliquez sur le bouton **importer le lutin depuis un fichier** (au-dessus de la liste des lutins) et sélectionnez l'image que vous venez d'enregistrer. Cela crée un nouveau lutin dont le costume est identique à l'image d'arrière-plan. Nommez-le Boule et éditez son costume dans l'éditeur d'images pour ne garder que la boule colorée dessinée sur la figure 3-14 (à gauche). Peignez-bien le contour de la boule avec la couleur transparente. Ensuite, placez le lutin Boule sur la scène, exactement au-dessus de l'endroit d'où il vient, afin qu'il semble faire partie de l'image d'arrière-plan (figure 3-11).



Figure 3-14 : Le costume du lutin Boule *tel qu'il apparaît dans l'éditeur d'images et son script*

La figure 3-14 montre également le script que vous devez ajouter au lutin Boule. Il change l'effet couleur du lutin continuellement pour donner l'illusion que les petits cercles changent réellement de couleur.

Créez le lutin Tableau de la même manière. La figure 3-15 montre l'apparence qu'il devrait avoir dans l'éditeur d'images (à gauche) et le script dont vous aurez besoin pour l'animer (à droite). J'ai ajouté quelques couleurs au costume (comparez avec la figure 3-11) pour faire en sorte que la commande **ajouter à l'effet couleur** fonctionne.

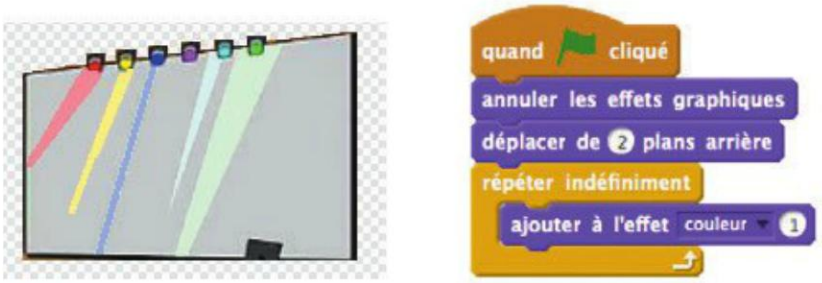


Figure 3-15 : Le lutin Tableau *et son script*

Le script renvoie Tableau deux plans en arrière afin que Danseur soit toujours à l'avant. Vous pourriez obtenir le même résultat en sélectionnant Danseur et en cliquant sur le bloc **envoyer au premier plan** de la palette *Apparence*.

Le dernier lutin de cette application est Projecteur. La figure 3-16 montre comment ce lutin apparaît dans l'éditeur d'images, ainsi que le script que vous devez créer. Le centre de l'image se trouve à la pointe du cône, qui représente le rayon lumineux.

Copyright © 2016 Eyrolles.

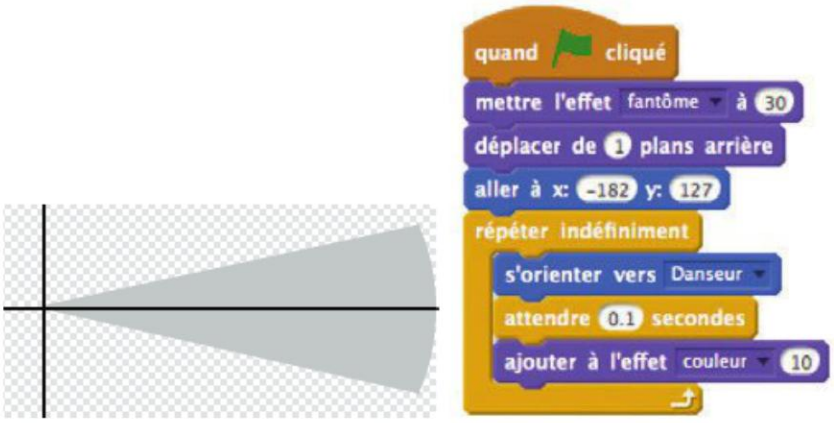


Figure 3-16 : Le lutin Projecteur *et son script*

Tout d'abord, le script règle l'effet fantôme du lutin à 30 pour le rendre transparent. Cela évite qu'il cache l'arrière-plan. Ensuite, le script envoie le lutin un plan en arrière, ce qui place le rayon lumineux derrière le danseur. Le lutin est ensuite positionné de façon à ce que le rayon de lumière semble sortir du projecteur (figure 3-10). Vous devez choisir les coordonnées *(x,y)* en vous basant sur votre dessin. Ceci fait, le script indique au rayon lumineux de suivre le danseur (en utilisant la commande **s'orienter vers**) et de changer sa couleur indéfiniment.

Quand vous aurez terminé l'ajout du projecteur, l'application devrait être complète. Cliquez sur le drapeau vert pour voir votre soirée dansante en action ! En plus de la musique et de la danse, vous devriez également voir les lutins Boule, Tableau et Projecteur changer de couleur comme de vraies lampes disco !

Feu d'artifice

FeudArtifice_PasDeCode.sb2

Une scène animée de feu d'artifice est une autre application qui se prête naturellement à l'utilisation des blocs graphiques et aux autres concepts abordés dans ce chapitre. Dans cette section, vous allez créer un feu d'artifice simple qui inondera le ciel d'éclats colorés. Les fusées exploseront à des instants aléatoires, produisant des éclats qui tomberont comme s'ils étaient attirés par la gravité et qui s'estomperont lentement (figure 3-17).

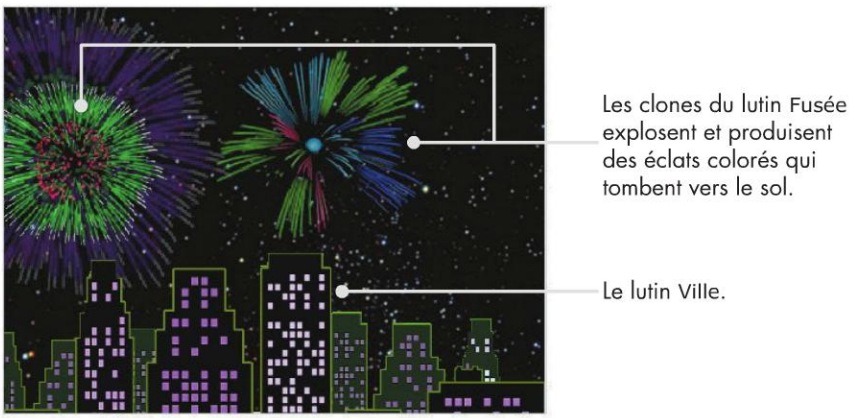


Figure 3-17 : L'animation de feu d'artifice en action

Copyright © 2016 Eyrolles.

Commencez par ouvrir le fichier *FeudArtifice_PasDeCode.sb2*, qui contient l'installation initiale de l'application, sans script. Cette dernière contient deux lutins : Ville et Fusée. Le lutin Ville est une image de grands immeubles que vous pourrez animer comme bon vous semblera. Le lutin Fusée créera en permanence des clones qui exploseront dans le ciel, produisant des feux d'artifice.

Le lutin Fusée possède huit costumes (figure 3-18). Le premier, *C1*, est un simple petit point que nous déploierons dans le ciel. Quand il atteindra sa destination, sélectionnée aléatoirement, il basculera vers l'un des autres costumes, également sélectionné aléatoirement, pour simuler l'explosion initiale. Nous utiliserons ensuite un effet graphique approprié pour rendre cette explosion plus réaliste.

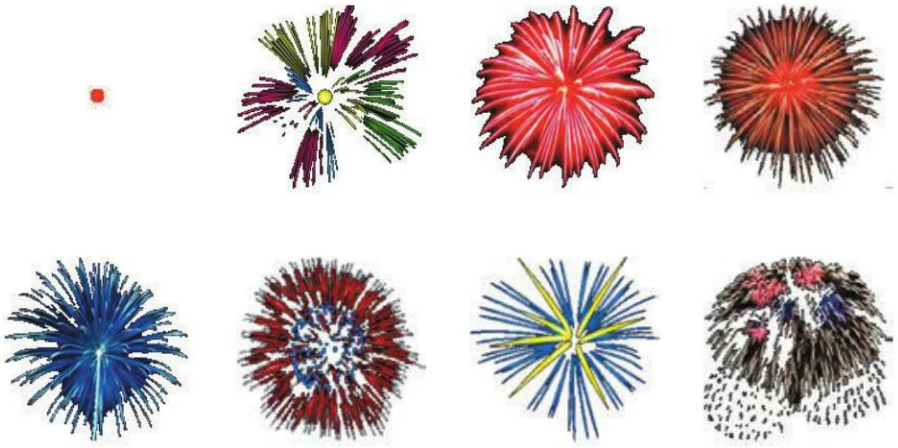


Figure 3-18 : Les huit costumes du lutin Fusée

Pour réaliser nos objectifs, commençons par écrire le script de la figure 3-19 pour le lutin Fusée. Ce script s'exécute lorsque l'utilisateur clique sur le drapeau vert. Après avoir caché le lutin Fusée, il démarre une boucle **répéter indéfiniment** qui crée des clones de lui-même à des instants aléatoires. Puisque ces clones héritent de l'état de visibilité de Fusée, tous sont initialement cachés.



Figure 3-19 : Le premier script du lutin Fusée

Nous devons à présent indiquer aux fusées clones ce qu'elles doivent faire (figure 3-20).

Copyright © 2016 Eyrolles.

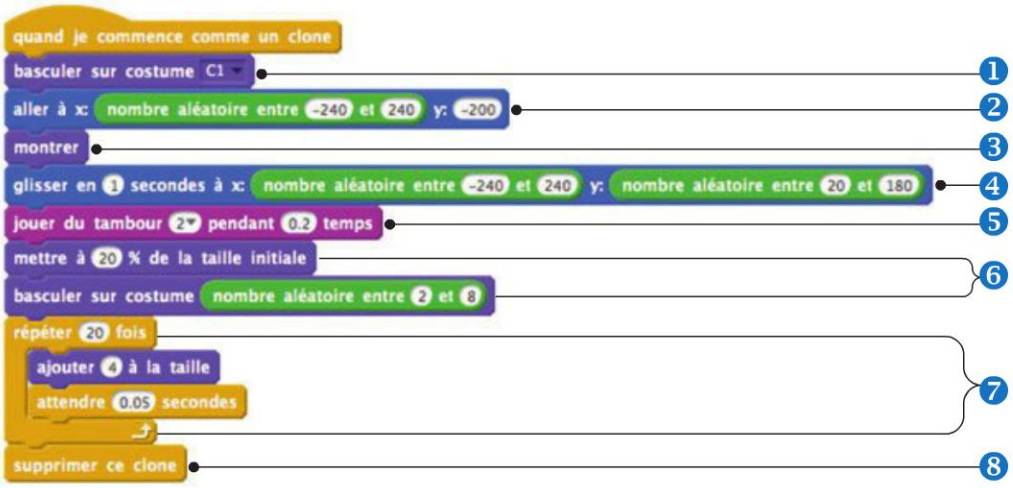


Figure 3-20 : Le script de démarrage des lutins clonés

Une fusée clonée commence par enfiler son premier costume ❶ (le petit point rouge). Elle se déplace ensuite à une position horizontale aléatoire en bas de la scène ❷, se montre ❸ et glisse vers une position aléatoire ❹ dans la partie supérieure de la scène (quelque part au-dessus des immeubles). Cette partie du script simule le lancement de la fusée et, si vous l'exécutez, vous voyez un petit point rouge se déplacer du sol au ciel. Quand le point atteint sa position finale dans le ciel, il explose grâce aux instructions de la seconde partie du script. Tout d'abord, le clone joue un petit son percussif ❺ (pour simuler le bruit d'une explosion). Puisque les explosions de feu d'artifice commencent petites avant de s'étendre, le clone règle sa taille à 20% et sélectionne l'un de ses autres costumes de façon aléatoire ❻. Il commence ensuite une boucle **répéter** ❼ pour faire grandir le feu d'artifice. À chaque passage dans la boucle, le clone augmente sa taille de 4. À la fin de la boucle, le clone se supprime ❽.

Et voilà votre feu d'artifice mis en place ! Vous devriez maintenant être capable d'exécuter une animation et de mettre en valeur la scène que vous avez créée. Avec seulement deux scripts, nous avons codé une animation relativement complexe.

Résumé

Nous avons présenté dans ce chapitre de nombreux nouveaux blocs de programmation qui pourront être utilisés pour donner du piquant à vos applications. Avec ces blocs, vous ajoutez de la couleur, de l'animation, des effets graphiques, de la musique et plus encore.

Nous avons étudié les blocs de la palette *Apparence* et avons donné plusieurs exemples sur la façon de les utiliser. Vous avez animé des lutins en les faisant changer de costumes. Vous avez pris connaissance des plans et avez vu comment ils affectent l'apparence des lutins qui se chevauchent.

Copyright © 2016 Eyrolles.

Nous avons ensuite fait le tour des commandes de la palette *Sons* et avons expliqué comment jouer des fichiers audio, des sons percussifs et des notes de musique. Vous avez créé une scène de danse complète avec les commandes des palettes *Apparence* et *Sons*, puis vous avez terminé en beauté avec une animation de feu d'artifice.

Dans le chapitre suivant, vous apprendrez comment coordonner les actions de différents lutins en utilisant l'envoi et la réception de messages. Vous apprendrez également comment décomposer un grand programme en petits morceaux plus faciles à gérer, appelés procédures. Ce concept est la clé pour écrire des applications plus complexes.

Problèmes

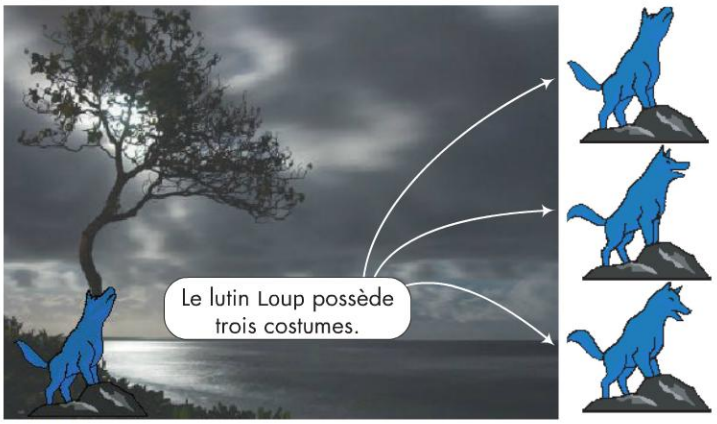
Zebre.sb2

1. Ouvrez l'application *Zebre.sb2*, présentée ci-après. Elle contient un lutin unique, Zèbre, qui possède trois costumes. Écrivez un script qui fasse se déplacer le zèbre sur la scène et changer ses costumes pour créer l'illusion qu'il galope.



Loup.sb2

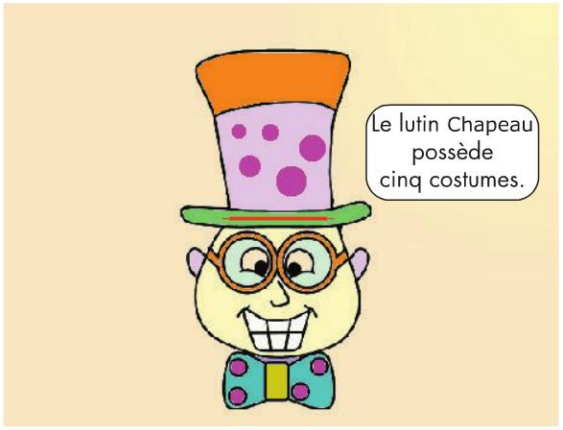
2. Ouvrez l'application *Loup.sb2*, présentée ci-après. Lorsque vous cliquez sur le drapeau vert, le lutin joue le son *WolfHowl*, qui dure environ 4 secondes. Créez le script qui synchronise le changement de costume du loup avec le son. Indice : insérez un bloc **attendre** avec un temps d'attente approprié après chaque changement de costume.



Copyright © 2016 Eyrolles.

ChangeDe Chapeau.sb2

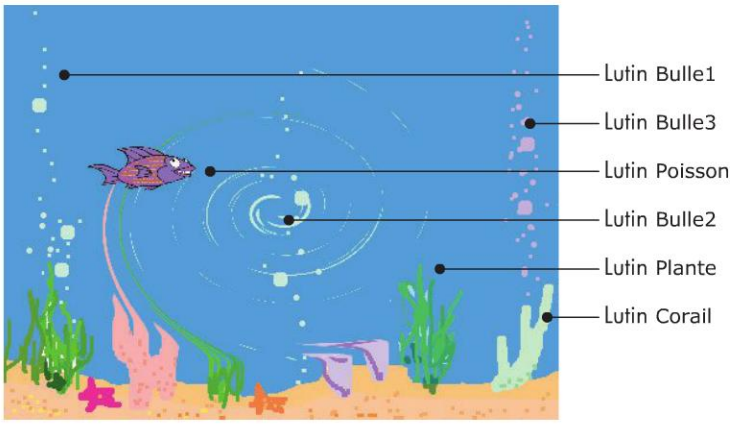
3. Ouvrez l'application *ChangeDeChapeau.sb2*, présentée ci-après. Le chapeau de cette application est un lutin qui possède cinq costumes. Créez le script qui le changera de costume à chaque fois qu'on clique dessus. Créez ensuite un jeu dans lequel le joueur habille un personnage en cliquant sur différents vêtements.



Aquarium.sb2

4. Ouvrez *Aquarium.sb2*. Cette application contient six lutins, comme illustré ci-après. Essayez différents effets graphiques pour animer l'aquarium. Voici quelques suggestions :

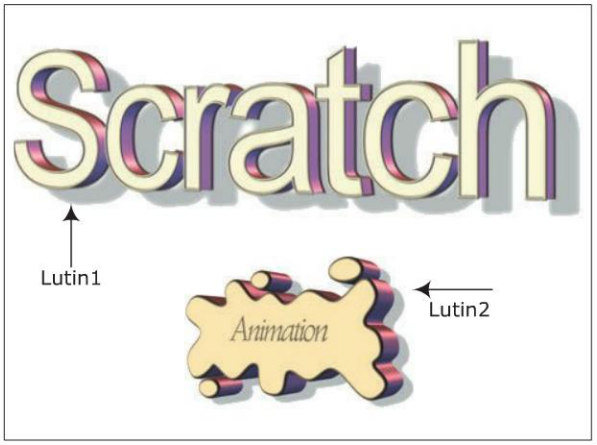
   – Utilisez l'effet tournoyer sur la scène. Commencez par un nombre élevé comme 1 000 pour créer un effet de vagues dans l'image de fond.
   – Changez les costumes des lutins Bulle1 et Bulle2 à un rythme approprié.
   – Déplacez le poisson sur la scène en le faisant changer de costume.
   – Appliquez l'effet fantôme au lutin Plante.
   – Utilisez l'effet couleur sur les lutins Corail et Bulle3.



Mots.sb2

5. Ouvrez l'application *Mots.sb2* et animez les mots en variant leur taille et leur orientation. Créez les deux scripts présentés sur la figure et exécutez l'application pour voir le résultat.

Copyright © 2016 Eyrolles.



Script de Lutin1



Script de Lutin2



Blague.sb2

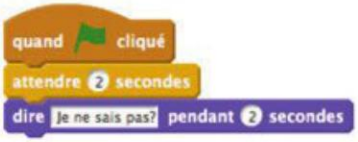
6. Ouvrez l'application *Blague.sb2*. Terminez les scripts afin que les lutins Garçon et Fille disent la blague de votre choix.



Script de Garçon



Script de Fille



Copyright © 2016 Eyrolles.

Nature.sb2

7. Ouvrez *Nature.sb2*. L'application contient trois lutins, comme le montre la figure ci-après. Animez la scène en utilisant les commandes de mouvement et de son. Voici quelques suggestions :
   - Le lutin Oiseau possède deux costumes qui donnent l'impression qu'il vole. Créez un script qui fera voler l'oiseau dans la scène et jouez le son *Oiseau* à des instants aléatoires.
   - Le lutin Canard possède 12 costumes qui lui font extirper un poisson hors de l'eau et le manger. Créez un script pour faire bouger le canard sur la scène et jouez le son *Canard* à des instants aléatoires.
   - Le lutin Phoque possède quatre costumes. Créez un script qui le fait jouer avec la balle et jouez le son *Phoque* à des instants aléatoires.



Copyright © 2016 Eyrolles.

4

LES PROCÉDURES

Ce chapitre explique comment programmer à la façon « diviser pour mieux régner ». Plutôt que de construire vos programmes en un seul gros morceau, vous écrirez des procédures séparées que vous assemblerez ensuite. L'utilisation des procédures rendra vos programmes plus simples à écrire, à tester et à déboguer. Dans ce chapitre, vous allez apprendre à utiliser :

- l'émission de messages pour coordonner le comportement de plusieurs lutins ;
- l'émission de messages pour implémenter des procédures ;
- la fonction de création de blocs personnalisés de Scratch 2 ;
- des techniques de programmation structurée.

Copyright © 2016 Eyrolles.

Pour la plupart, les applications que nous avons développées jusqu'à présent ne contenaient qu'un seul lutin, alors que les programmes plus complexes nécessitent plusieurs lutins qui fonctionnent ensemble. Une histoire animée, par exemple, peut contenir plusieurs personnages ainsi que plusieurs arrière-plans. Nous avons donc besoin d'un outil pour synchroniser les actions des différents lutins. Dans ce chapitre, nous utiliserons le mécanisme d'émission de messages de Scratch pour coordonner les actions (c'était la seule manière d'implémenter des procédures dans la version précédente du logiciel). Nous verrons ensuite comment utiliser la fonction de « blocs personnalisés » pour structurer de longs programmes en de plus petites parties, plus simples à gérer, appelées *procédures*. Une procédure est une séquence de commandes qui exécute une fonction spécifique. Par exemple, nous pouvons créer des procédures qui feront dessiner des formes à un lutin, accomplir des calculs complexes, traiter les actions de l'utilisateur, jouer une séquence de notes musicales, gérer des jeux, ou bien d'autres choses encore. Une fois créées, ces procédures peuvent servir de blocs de base pour créer toutes sortes d'applications utiles.

Émission et réception de messages

Comment fonctionne le système d'émission de messages de Scratch en pratique ? Chaque lutin est capable d'envoyer un message (nommé comme vous le souhaitez) à l'aide des blocs **envoyer à tous** et **envoyer à tous et attendre** (de la palette *Événements*) présentés à la figure 4-1. Cet envoi de message déclenche tous les scripts de tous les lutins qui commencent avec le bloc de déclenchement **quand je reçois** correspondant (y compris pour le lutin qui envoie le message). Tous les lutins entendent ce message, mais ils ne réagissent que s'ils possèdent le bloc **quand je reçois** correspondant.

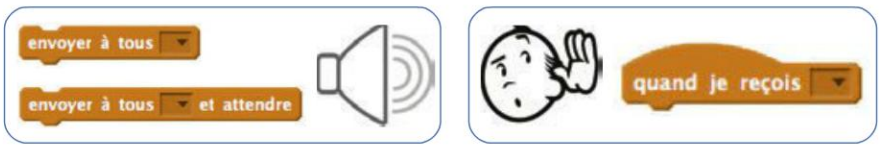


Figure 4-1 : Vous pouvez utiliser les blocs d'envoi et de réception de messages pour coordonner les actions de plusieurs lutins.

Considérez la figure 4-2. Elle montre quatre lutins : étoile de mer, chat, grenouille et chauve-souris. L'étoile de mer envoie le message sauter à tous les lutins, y compris elle-même. En réponse à ce message, le chat et la grenouille exécutent leur script sauter. Notez qu'ils sautent chacun à sa manière, car ils exécutent un script différent. La chauve-souris reçoit également le message, mais elle n'y réagit pas car on ne lui a pas dit de faire quoi que ce soit à la réception de ce message. Le chat de cette figure sait comment marcher et sauter, la grenouille ne peut que sauter et la chauve-souris ne peut que voler.

La commande **envoyer à tous et attendre** fonctionne de la même manière que **envoyer à tous** mais, avant de passer à l'instruction suivante, elle attend que tous les lutins aient terminé l'exécution de tous les blocs qui répondent au message envoyé.

Copyright © 2016 Eyrolles.

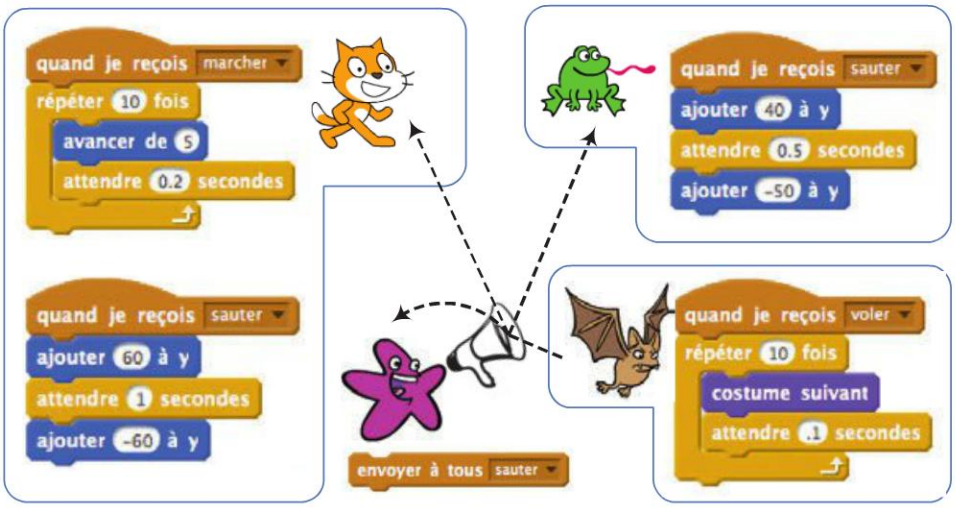


Figure 4-2 : Un message est reçu par tous les lutins, même par celui qui l'a envoyé.

Envoyer et recevoir des messages

AppliCarre.sb2

Pour comprendre le fonctionnement de l'émission et de la réception des messages, créons une application simple qui dessine des carrés de couleur aléatoire. Quand l'utilisateur clique avec le bouton gauche de la souris sur la scène, celle-ci détecte un événement (en utilisant le bloc **quand la scène est cliquée** de la palette *Événements*) et envoie un message appelé Carré (vous pouvez choisir un autre nom si vous le désirez). Lorsque l'unique lutin de cette application reçoit ce message, il se déplace à la position actuelle de la souris et dessine un carré. Suivez les étapes ci-après pour créer l'application :

1. Démarrez Scratch et sélectionnez **Nouveau** dans le menu **Fichier** pour démarrer une nouvelle application. Vous pouvez changer le costume du chat si vous le désirez.
2. Ajoutez le bloc **quand je reçois** (de la palette *Événements*) dans la zone de scripts du lutin. Cliquez sur la petite flèche noire de ce bloc et sélectionnez **nouveau message…** à partir du menu déroulant. Dans la fenêtre qui apparaît, tapez Carré et cliquez sur OK. Le nom du bloc devrait être à présent **quand je reçois Carré**.
3. Complétez le script comme le montre la figure 4-3. Tout d'abord, le lutin soulève son stylo et se déplace vers la position actuelle de la souris, indiquée par les blocs **souris x** et **souris y** (de la palette *Capteurs*). Ensuite, il sélectionne une couleur aléatoire, met le stylo en position d'écriture et dessine un carré.

Le lutin est maintenant prêt à réagir au message Carré lorsqu'il le recevra. Le script de la figure 4-3 peut être appelé *script de gestion de messages*, puisque sa fonction est de gérer (ou de traiter) la réception d'un message.

Copyright © 2016 Eyrolles.

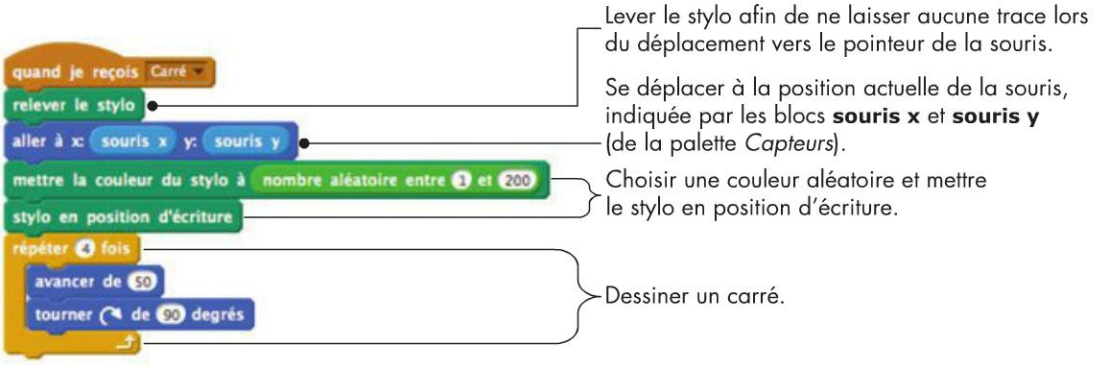


Figure 4-3 : Le script de gestion du message Carré

À présent, faisons en sorte qu'un clic sur la scène déclenche l'émission d'un message Carré. Pour ce faire, cliquez sur la miniature de la scène dans la liste des lutins et ajoutez-lui les scripts de la figure 4-4. Le premier efface de la scène tous les précédents tracés du stylo lorsqu'on clique sur le drapeau vert. Le second script, qui est déclenché lorsque l'utilisateur clique sur la scène, utilise le bloc **envoyer à tous** pour indiquer au lutin qu'il est temps de dessiner.



Figure 4-4 : Les deux scripts de la scène pour l'application de dessin de carré.

L'application est maintenant complète. Pour la tester, cliquez simplement sur la scène. Chaque clic devrait déclencher le dessin d'un carré.

L'envoi de messages pour la coordination de plusieurs lutins

Fleurs.sb2

Pour voir plusieurs lutins répondre au même message, créons une application qui dessine plusieurs fleurs lorsqu'on clique sur la scène. L'application Fleurs contient cinq lutins (nommés Fleur1 à Fleur5), chacun responsable d'un dessin sur la scène. Chaque lutin a son propre costume (figure 4-5) dont le fond est transparent. Observez-bien l'emplacement du centre de rotation de chaque costume (au croisement des lignes noires).

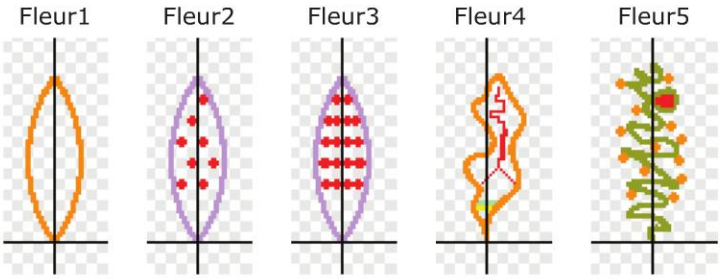


Figure 4-5 : L'application utilise ces 5 lutins pétales (présents dans l'éditeur d'images).

Copyright © 2016 Eyrolles.

Lorsqu'un lutin reçoit un message lui indiquant de dessiner sa fleur, il estampille sur la scène plusieurs copies pivotées de son costume. La figure 4-6 montre également quelques résultats obtenus avec l'application de dessin de fleurs que nous allons explorer ensuite.

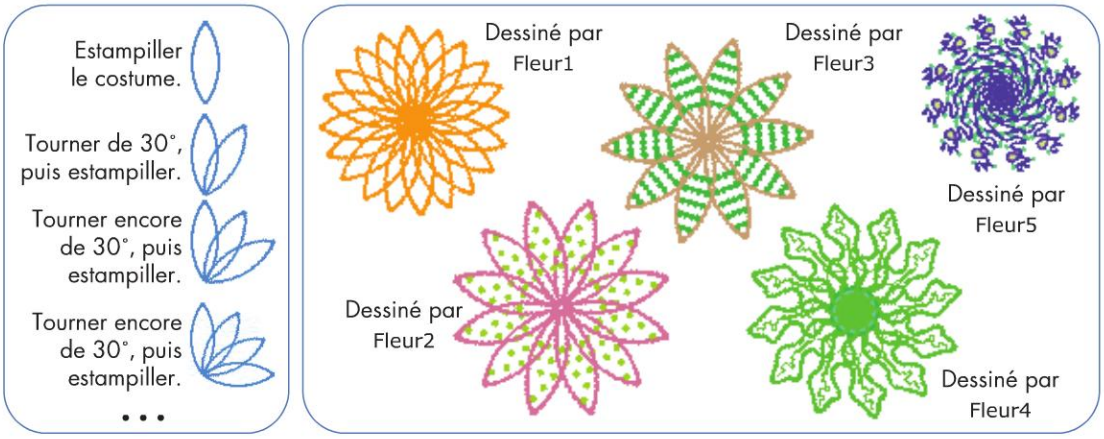


Figure 4-6 : La procédure de dessin de l'application (à gauche) et quelques exemples de fleurs (à droite)

Lorsque vous cliquez sur la scène, celle-ci le détecte grâce au bloc **quand la scène est cliquée**. Elle efface alors son arrière-plan et envoie un message appelé Dessin. Tous les lutins répondent à ce message en exécutant un script similaire à celui présenté à la figure 4-7.

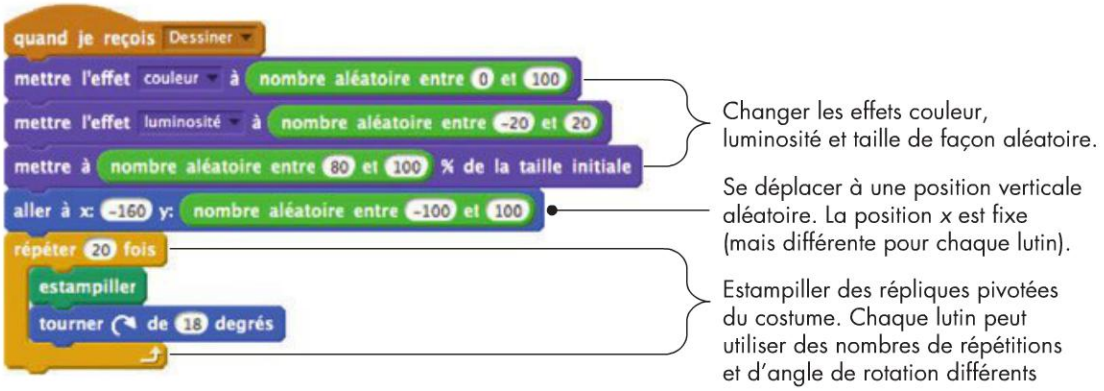


Figure 4-7 : Le script de base utilisé pour chaque lutin

Le script commence par affecter des valeurs aléatoires aux effets couleur, luminosité et taille pour changer l'apparence de la fleur dessinée. Il déplace ensuite le lutin à une position verticale aléatoire sur la scène et dessine une fleur en estampillant puis en pivotant des répliques de son costume.

Ouvrez cette application (nommée *Fleurs.sb2*) et exécutez-la pour voir comment elle fonctionne. Malgré sa simplicité, son résultat est intéressant. Je vous encourage à concevoir différents costumes et à changer le centre de rotation pour découvrir de nouveaux concepts de fleurs.

Maintenant que vous avez compris comment envoyer et recevoir des messages, je vais vous présenter la programmation structurée, qui vous permettra de gérer la complexité de longs programmes.

Copyright © 2016 Eyrolles.

Créer de longs programmes en petites étapes

Les scripts que vous avez écrits jusqu'à présent sont relativement courts et simples. Vous écrirez sans doute des scripts plus complexes qui contiendront des centaines de blocs. Leur compréhension et leur maintenance deviendront alors des tâches très difficiles.

Une approche appelée *programmation structurée* a été développée au milieu des années 1960 pour simplifier l'écriture, la compréhension et la maintenance de programmes informatiques. Au lieu d'écrire un seul long programme, cette approche consiste à diviser le code en petits morceaux, qui résoudront chacun une partie de la tâche générale.

Considérez, par exemple, la préparation d'un gâteau. Vous ne pensez sûrement pas à chaque étape individuelle lorsque vous cuisinez, mais la procédure suit une recette précise qui liste les étapes nécessaires. Celle-ci peut contenir des instructions telles que (1) mélangez 4 œufs, 50 g de farine et un verre d'eau, (2) versez le mélange dans un moule, (3) placez le moule dans le four, (4) laissez cuire 1 h à 180 °C, et ainsi de suite. Par essence, la recette divise de façon logique la préparation du gâteau en étapes distinctes.

De même, lorsque vous concevez une solution pour répondre à un problème de programmation, le fait de décomposer le problème en petites parties vous sera d'une grande aide. Cette approche vous donne une bonne vision globale du programme et de la façon dont interagissent ses différentes parties.

Considérez la figure 4-8, qui montre un long script dessinant une forme sur la scène. Vous verrez que vous pouvez diviser ce script en plusieurs fonctions plus petites. Les six premiers blocs, par exemple, initialisent le lutin. Le premier bloc **répéter** dessine un carré, le second dessine un triangle, etc. En utilisant la programmation structurée, nous pouvons regrouper les blocs qui exécutent une tâche commune et former ainsi des procédures qui porteront des noms représentatifs.

Une fois ces procédures écrites, nous pouvons les appeler en séquence pour résoudre notre problème de programmation. La figure 4-8 montre également comment les procédures sont assemblées pour effectuer la même fonction que celle du script original. Le script qui utilise les procédures (à droite) est plus modulaire et plus simple à comprendre que le programme original (à gauche).

Les procédures évitent également d'écrire plusieurs fois le même code. Si un ensemble de commandes est exécuté à plusieurs reprises dans un programme, utilisez à la place une procédure qui regroupe ces commandes. Cette stratégie est appelée *réutilisation de code*. Par exemple, regardez comment la procédure **Dessiner un carré** a été réutilisée sur la figure 4-8.

Les procédures permettent d'utiliser la stratégie « diviser pour mieux régner » pour résoudre des problèmes complexes. Vous divisez des problèmes longs et complexes en sous-problèmes et vous régnez ensuite sur ces problèmes individuellement, en les testant chacun de manière isolée. Après avoir résolu tous les sous-problèmes un à un, vous assemblez ces pièces de façon à ce qu'elles résolvent le problème original. Ceci est similaire à notre stratégie de préparation de gâteau : notre recette divisait le problème en étapes bien définies et nous avons exécuté chaque étape dans un ordre correct pour construire notre projet final (notre gâteau).

Copyright © 2016 Eyrolles.

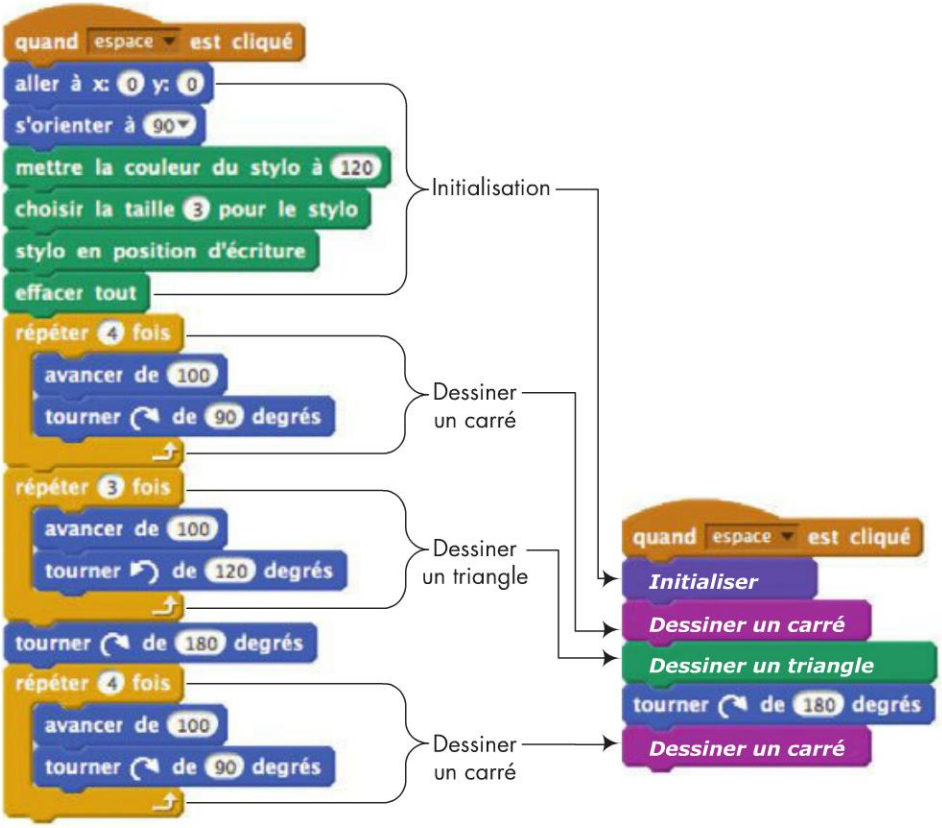


Figure 4-8 : Diviser un long script en parties logiques qui réalisent chacune une fonction.

Vous vous demandez sans doute comment créer ces procédures ? Avant Scratch 2, il était impossible de construire le bloc **Initialisation** de la figure 4-8 et de l'appeler dans votre script. La seule façon de simuler l'utilisation des procédures et de structurer les programmes de Scratch était d'utiliser le mécanisme d'*envoi de messages*. Dans Scratch 2 toutefois, la fonction puissante de blocs personnalisés a été ajoutée.

Dans cette section, nous montrerons comment utiliser l'ancienne méthode, puisque vous la retrouverez dans les programmes créés avec une version antérieure de Scratch. La fonction de création de blocs personnalisés sera expliquée dans la section suivante et sera régulièrement utilisée dans le reste du livre.

Puisque les lutins reçoivent leurs propres émissions de messages, nous pouvons implémenter des procédures en faisant en sorte qu'un lutin s'envoie un message à lui même et qu'il exécute alors la tâche désirée sous un bloc de déclenchement **quand je reçois**. Nous pouvons utiliser le bloc **envoyer à tous et attendre** pour nous assurer que nos procédures sont appelées dans le bon ordre, ce qui ajoutera structure et modularité à nos programmes.

Créer des procédures avec l'émission de messages

Fleurs2.sb2

Nous allons expliquer comment fonctionnent les procédures et comment elles peuvent améliorer votre code en recréant le programme Fleurs que nous avons vu précédemment.

Ouvrez le fichier *Fleurs2.sb2*, qui contient la nouvelle version du programme. Le script de la scène est le même que précédemment (envoi d'un

Copyright © 2016 Eyrolles.

message Dessiner à chaque clic) mais, cette fois-ci, notre programme n'utilise qu'un seul lutin au lieu de cinq. Ce dernier possède cinq costumes, pétale1 à pétale5, et appelle une procédure qui dessine une fleur pour chaque costume. Puisque nous n'avons qu'un seul lutin, nous n'avons besoin que d'une version du code de dessin (et non pas cinq comme dans la première version). Cela rétrécit le programme et facilite la compréhension du code. Dans cette application, lorsque le lutin reçoit le message Dessiner, il exécute le script présenté sur la figure 4-9.

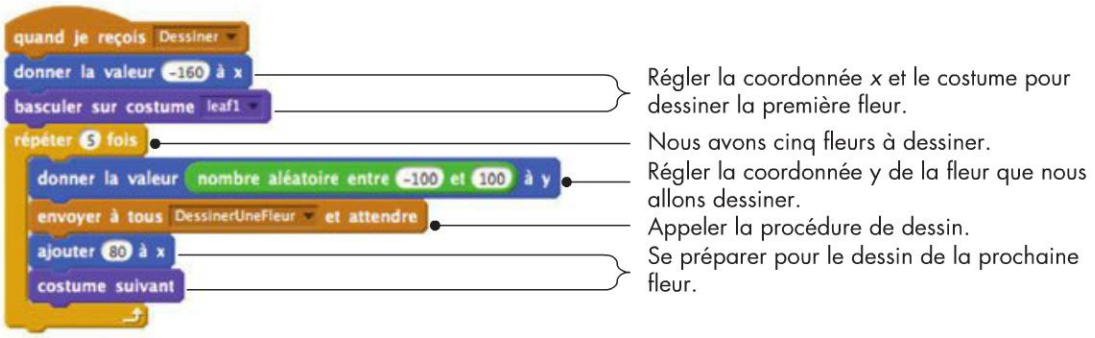


Figure 4-9 : Lorsque le lutin reçoit le message Dessiner, *il appelle* **DessinerUneFleur** *cinq fois (dans une boucle) pour tracer cinq fleurs.*

Le script règle la coordonnée *x* et le costume pour dessiner la première fleur, puis entre dans une boucle pour dessiner cinq fleurs. À chaque passage, la boucle règle la coordonnée *y* et appelle DessinerUneFleur en envoyant un message. Ceci arrête l'exécution du script jusqu'à ce que le tracé soit terminé. Le script Dessiner redémarre ensuite son exécution, en ajustant la coordonnée *x* et en changeant de costume pour se préparer au dessin de la fleur suivante.

NOTE *Vous pouvez nommer une procédure comme bon vous semble, mais je recommande le choix d'un nom qui en reflète le but. Cela vous aidera, particulièrement lorsque vous retournerez voir un programme que vous avez créé des mois auparavant. Par exemple, si vous voulez montrer aux joueurs combien ils ont gagné de points, vous pouvez créer une procédure appelée* MontrerLeScore. *En nommant cette procédure* Marie *ou* Alphonse, *vous ne vous rappellerez sans doute pas pourquoi vous l'aviez créée (et il en sera de même pour quiconque tentera de lire votre programme).*

La procédure DessinerUneFleur est représentée à la figure 4-10. Elle donne des valeurs aléatoires aux effets couleur, luminosité et taille avant d'estampiller des versions pivotées du costume actuel pour dessiner une fleur.

Alors que la première version du programme contenait cinq lutins et cinq scripts répétés, la seconde version donne le même résultat avec un seul lutin qui appelle une procédure pour dessiner les cinq fleurs. Ouvrez *Fleurs.sb2* et *Fleurs2.sb2* dans deux onglets de votre navigateur pour les comparer. La nouvelle version n'est-elle pas bien plus simple à suivre ? L'utilisation de procédures vous permet d'écrire des programmes plus petits et plus simples à comprendre et à maintenir. Cela vous sera de plus en plus utile lorsque vous créerez des programmes complexes.

Copyright © 2016 Eyrolles.

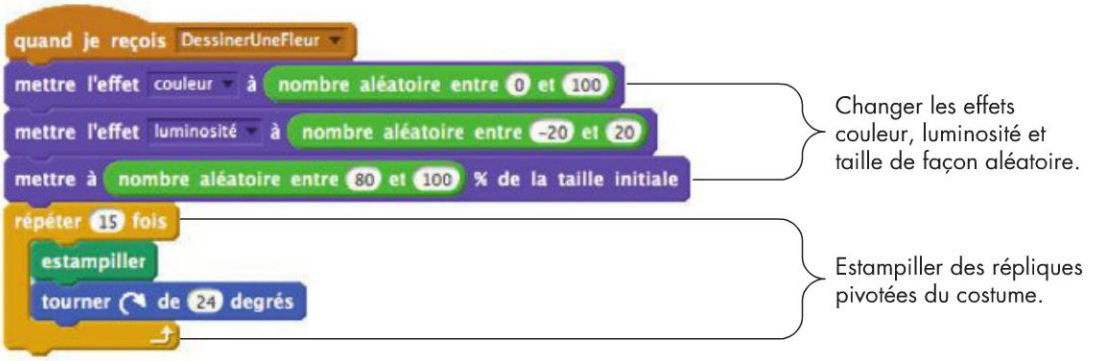


Figure 4-10 : La procédure **DessinerUneFleur**

Construire vos propres blocs

Dans Scratch 2, vous pouvez également créer votre propre bloc. Celui-ci devrait apparaître dans la palette *Ajouter blocs*, où vous pourrez l'utiliser de la même manière que ceux de Scratch.

Nous allons modifier le programme Fleurs2 de la section précédente en utilisant un bloc personnalisé pour la procédure DessinerUneFleur :

1. Tout d'abord, ouvrez le fichier *Fleurs2.sb2* que nous avons utilisé dans la section précédente. Sélectionnez **Fichier>Télécharger dans votre ordinateur** et sauvegardez-le en le nommant *Fleurs3.sb2*.
2. Cliquez sur la miniature du lutin Fleur pour le sélectionner. Allez ensuite dans la palette *Ajouter blocs* et cliquez sur **Créer un bloc**. Vous devriez voir la fenêtre présentée sur la figure 4-11 (à gauche). Tapez DessinerUneFleur dans le nom du bloc et cliquez sur **OK**. Un nouveau bloc de fonction appelé **DessinerUneFleur** devrait apparaître dans la palette *Ajouter blocs*, ainsi qu'un bloc **définir DessinerUneFleur** dans la zone de script du lutin (figure 4-11, à droite).

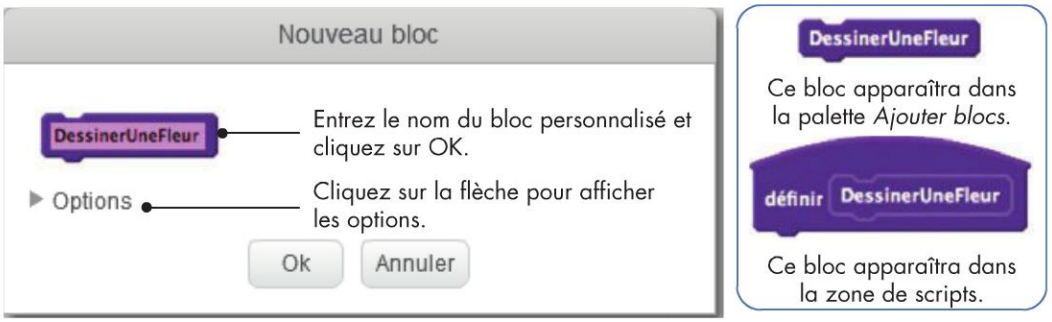


Figure 4-11 : La fenêtre Nouveau Bloc et les blocs qui apparaissent après avoir créé le bloc **DessinerUneFleur**

Copyright © 2016 Eyrolles.

3. Détachez le script connecté au bloc **quand je reçois DessinerUneFleur** (vous pouvez supprimer ce dernier puisque vous n'en aurez plus besoin) et connectez-le au bloc **définir DessinerUneFleur** (figure 4-12). Cela définit une nouvelle procédure, appelée **DessinerUneFleur**, implémentée en tant que bloc personnalisé.

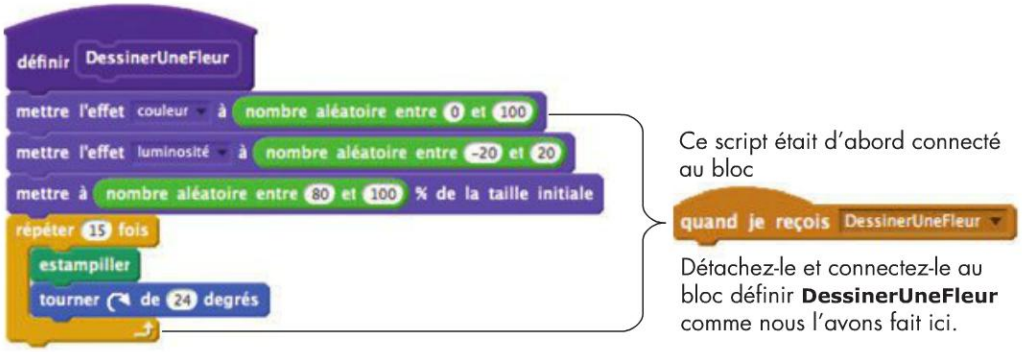


Figure 4-12 : La procédure **DessinerUneFleur** *implémentée en tant que bloc personnalisé*

4. Maintenant que la procédure **DessinerUneFleur** a été créée, il n'y a plus qu'à l'appeler dans le script de gestion de messages Dessiner. Modifiez ce script comme le montre la figure 4-13. Notez que nous avons simplement remplacé **envoyer à tous DessinerUneFleur et attendre** par le bloc personnalisé **DessinerUneFleur**.

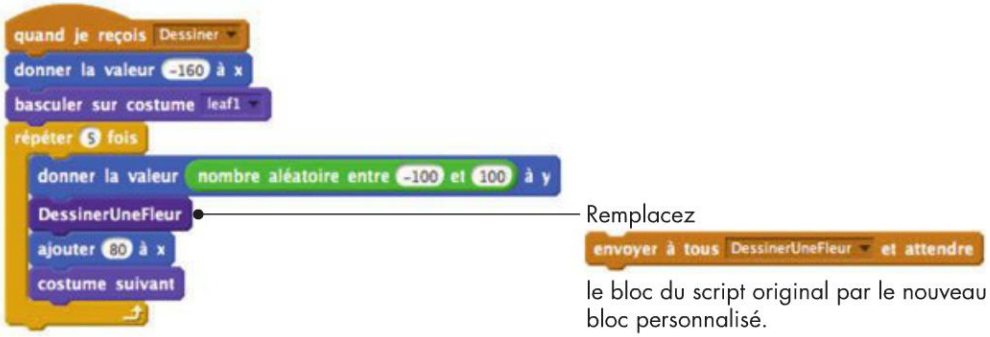


Figure 4-13 : Appel de la procédure **DessinerUneFleur** *dans le script de gestion de messages* Dessiner.

Le programme est maintenant complet et vous pouvez le tester. Cliquez sur la scène pour vérifier qu'il fonctionne encore comme avant.

Maintenant que vous connaissez les principes fondamentaux des blocs personnalisés, vous pouvez aller un peu plus loin en créant des blocs qui acceptent les entrées.

Copyright © 2016 Eyrolles.

EXÉCUTER SANS RAFRAÎCHISSEMENT DE L'ÉCRAN

L'implémentation de la procédure **DessinerUneFleur** avec les blocs personnalisés apporte une nouvelle fonction qui peut raccourcir la durée d'exécution d'un script de dessin. Pour le démontrer, suivez les étapes ci-après :

1. Cliquez droit sur le bloc **DessinerUneFleur** de la palette *Ajouter blocs* et sélectionnez **éditer** dans le menu contextuel. Cela devrait ouvrir la fenêtre de la figure 4-11, avec le titre pour seule différence, qui est passé de *Nouveau bloc* à *Éditer un bloc*.
2. Cliquez sur la flèche à gauche de Options et cochez la case qui accompagne **Exécuter sans rafraîchissement de l'écran**, puis cliquez sur **OK** (figure 4-15).
3. Cliquez à présent sur la scène. Au lieu de voir les pétales s'estampiller et pivoter un à un pour chaque fleur, vous devriez les voir apparaître sur la scène *presque* en même temps.

La procédure **DessinerUneFleur** contient plusieurs blocs qui modifient l'apparence d'un lutin, dont **mettre l'effet couleur à**, **mettre l'effet luminosité à**, **mettre l'effet taille à** et **estampiller**. Après avoir exécuté un bloc de ce type, Scratch effectue normalement une pause pour *rafraîchir* (ou redessiner) l'écran. Voilà pourquoi vous pouviez voir l'évolution des dessins lorsque l'application s'exécutait.

Si vous sélectionnez l'option **Exécuter sans rafraîchissement de l'écran**, les blocs s'exécutent sans pause, donc plus rapidement. L'écran se rafraîchira une fois que Scratch aura réalisé la procédure complète.

En plus d'accélérer une procédure, l'option **Exécuter sans rafraîchissement de l'écran** évite le tremblement de l'écran qui peut être induit par des rafraîchissements répétés.

Transmettre des paramètres aux blocs personnalisés

Commençons par créer un bloc personnalisé nommé **Carré**, qui dessine un carré dont le côté fait 100 pixels de long (figure 4-14).



Figure 4-14 : Une procédure **Carré** *qui dessine un carré de taille fixe*

Copyright © 2016 Eyrolles.

Cette procédure a des capacités limitées, car la taille du carré a été fixée une bonne fois pour toutes. Comment faire si vous voulez dessiner des carrés de côtés 50, 75 ou 200 pixels ? Vous pourriez définir plusieurs blocs personnalisés nommés **Carré50**, **Carré75** et **Carré200** mais, la plupart du temps, déclarer des blocs qui font quasiment la même chose n'est pas une bonne idée : si vous devez changer une partie du script de la procédure, alors vous devrez le faire dans toutes les copies. Il existe une meilleure solution : utiliser un seul bloc personnalisé **Carré** qui permette à l'utilisateur de spécifier la taille désirée au moment où ce bloc est utilisé.

Vous avez déjà appliqué ce concept. Par exemple, Scratch fournit un unique bloc **avancer de** qui vous demande de spécifier de combien de pas le lutin se déplacera, en entrant un nombre dans la case de paramètre du bloc. Ainsi, Scratch n'a pas besoin de fournir autant de blocs que de distances de déplacement possibles.

Nous devons donc à présent ajouter une case de paramètre à notre bloc **Carré**, dans laquelle l'utilisateur pourra rentrer la longueur des côtés du carré (figure 4-15).

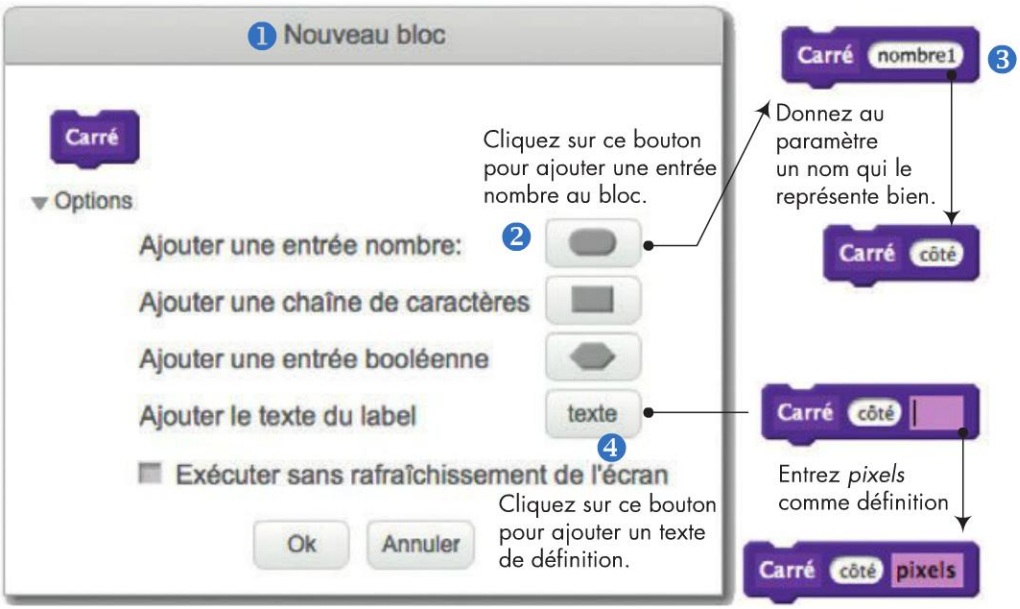


Figure 4-15 : Ajout d'une entrée nombre au bloc **Carré**

Cliquez droit sur le bloc **Carré** de la palette *Ajouter blocs* (ou sur le bloc **définir Carré** de la zone de scripts) et sélectionnez **éditer** dans le menu contextuel pour faire apparaître la fenêtre Éditer un bloc ❶. Cliquez sur la flèche à gauche de **Options**. Nous souhaitons que notre bloc **Carré** prenne en entrée la longueur du côté, qui est un nombre. Cliquez donc sur **Ajouter une entrée nombre** ❷. Une case nommée number1 s'ajoute au bloc.

Scratch ne sait que faire du nom que vous donnerez à ce paramètre, mais ce n'est pas votre cas ! Donnez-lui un nom qui ait du sens ❸, tel que côté, longueur, ou longueurDeCôté. Utilisons le nom côté pour cet exemple.

Techniquement, c'est tout ce que vous aurez à faire pour ajouter une entrée nombre à votre procédure. Si nous cliquons sur **OK**, nous sauvegardons le bloc **Carré** qui prend un nombre en entrée. Nous pourrions donc

Copyright © 2016 Eyrolles.

faire glisser ce bloc dans notre script, préciser la longueur désirée dans la case de paramètre et obtenir ainsi **Carré 50**. Cependant, comment l'utilisateur peut-il savoir ce que ce nombre signifie ? Est-ce une aire de 50, une diagonale de 50, un côté de 50 ou autre chose ?

Imaginez que le bloc **glisser** de Scratch ressemble à ceci :



Comment pourriez-vous savoir que la première case représente un temps en secondes et que les deux autres représentent les coordonnées *x* et *y* du point cible vers lequel le lutin doit glisser ? Les concepteurs de Scratch ont eu la présence d'esprit de rendre le bloc **glisser** plus simple à comprendre et à utiliser en ajoutant du texte de définition de la façon suivante :



Réalisons la même chose avec notre bloc **Carré** en ajoutant du texte qui décrive le sens (ou l'utilisation) de la case de paramètre. Cliquez sur **Ajouter le texte du label** ❹ (figure 4-15) pour ajouter un texte après le paramètre côté. Entrez-y **pixels** et cliquez sur **OK**.

À présent, si vous observez la définition de la procédure **Carré** dans la zone de scripts, vous voyez un petit bloc (nommé côté) ajouté à l'en-tête (figure 4-16, à gauche). Le bloc **avancer de** contient toujours le nombre fixe 100 que nous avions entré précédemment, mais nous n'avons plus qu'à faire glisser le bloc côté de l'en-tête de la méthode **Carré** et à le déposer sur la case de paramètre du bloc **avancer de** pour remplacer ce nombre (figure 4-16, à droite).

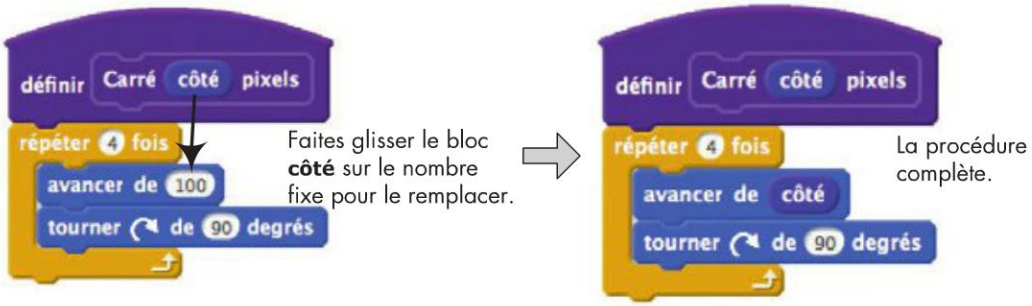


Figure 4-16 : Modification de la procédure **Carré** *pour utiliser le paramètre* côté

Le bloc **côté**, qui apparaît dans l'en-tête de la procédure **Carré**, est appelé *paramètre*. Nous voulions que notre procédure soit capable de dessiner un carré de n'importe quelle taille. Donc, au lieu de programmer en dur un nombre fixe à l'intérieur de plusieurs procédures, nous avons utilisé un paramètre générique appelé côté. Les utilisateurs pourront spécifier la valeur exacte de côté quand ils appelleront la procédure. Illustrons ce point en modifiant le script de la figure 4-14 pour obtenir une nouvelle version de notre procédure **Carré** (figure 4-17).

Copyright © 2016 Eyrolles.

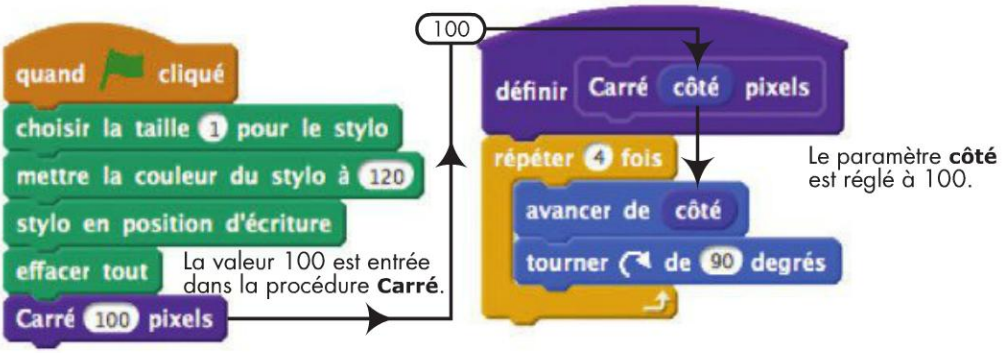


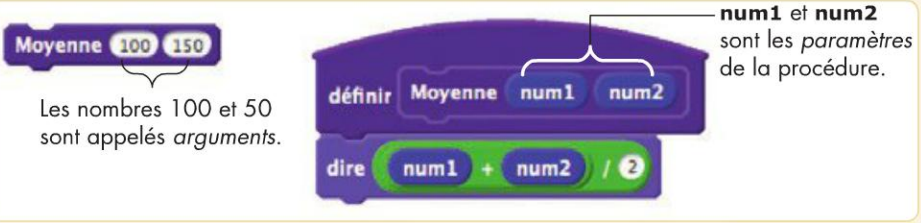
Figure 4-17 : Appel de la procédure **Carré** *avec* côté *réglé à 100*

Ici, le nombre 100 (appelé *argument*) est envoyé à la procédure **Carré**. Quand celle-ci s'exécute, la valeur 100 est utilisée par toutes les occurrences du bloc côté. Pouvoir spécifier différents arguments dans une procédure est une fonction puissante qui ajoute beaucoup de flexibilité à vos programmes.

Nous pouvons encore améliorer notre procédure en faisant en sorte qu'elle prenne la couleur du carré comme second paramètre (figure 4-18). Ici, nous avons ajouté un deuxième paramètre d'entrée, nommé numClr (abréviation de numéro de couleur), qui indique la couleur désirée pour le carré. La procédure règle à présent la couleur du stylo à la valeur spécifiée par numClr avant d'exécuter la boucle de dessin. Éditez le bloc **Carré** pour implémenter les changements montrés sur la figure.

PARAMÈTRE OU ARGUMENT ?

Bien que de nombreux programmeurs utilisent *paramètre* et *argument* pour désigner la même chose, ces deux termes sont en fait différents. Pour clarifier, considérez la procédure **Moyenne** présentée ci-après, qui calcule la moyenne de deux nombres.



Cette procédure possède deux paramètres nommés num1 et num2. Un *paramètre* définit l'entrée d'une procédure. Cette procédure peut être appelée avec le bloc présenté à droite, dans lequel peuvent être spécifiées des valeurs ou des expressions dans les cases disponibles. Les valeurs 100 et 50 de l'exemple sont appelées *arguments* de la procédure.

Le nombre d'arguments dans l'appel correspond au nombre de paramètres dans la définition de la procédure. Lorsque vous appelez **Moyenne**, les paramètres num1 et num2 reçoivent respectivement les valeurs 100 et 50, puisque les arguments et les paramètres ont des positions correspondantes.

Copyright © 2016 Eyrolles.

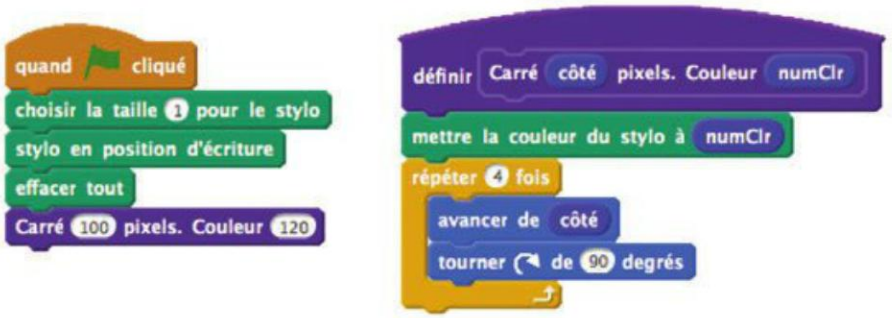


Figure 4-18 : Cette version de **Carré** *prend comme second paramètre d'entrée la couleur désirée.*

FAITES LE TEST 4-1

Pourquoi ne pas ajouter l'épaisseur des contours du carré comme paramètre d'entrée ? Modifiez la procédure **Carré** pour prendre ce troisième paramètre, appelé `tailleStylo`, qui spécifiera la taille du stylo à utiliser lors du dessin.

Pour conclure cette section, je vous propose quelques astuces utiles à la construction de blocs personnalisés.

- Les blocs personnalisés ne peuvent pas être partagés d'un lutin à un autre. Par exemple, si vous créez un bloc personnalisé pour `Lutin1`, alors seul `Lutin1` pourra l'utiliser. De même, un bloc défini pour la scène ne pourra être utilisé que dans les scripts appartenant à cette dernière.
- Donnez à vos paramètres des noms clairs qui indiqueront ce pour quoi ils sont utilisés.
- Pour supprimer un bloc personnalisé, vous n'avez qu'à faire glisser son bloc **définir** (son bloc chapeau) de la zone de scripts vers celle des palettes. Cela ne sera possible que si votre projet ne contient aucun script associé. Éliminez donc le bloc personnalisé de vos scripts avant d'essayer de le supprimer.
- Pour supprimer un paramètre d'un bloc personnalisé, cliquez sur son nom dans la fenêtre Éditer un bloc, puis sur la croix qui apparaît au-dessus de sa case.
- En plus des entrées nombres, vous pouvez ajouter des chaînes de caractères et des paramètres booléens. Nous étudierons ces types de données plus en détail lorsque nous parlerons des variables dans le chapitre suivant.

À présent, vous vous demandez peut-être si une procédure peut en appeler une autre ? Dans la section suivante, vous apprendrez comment utiliser des appels imbriqués pour étendre la puissance et l'utilité de procédures existantes.

Utiliser des procédures imbriquées

Une procédure devrait être conçue pour n'accomplir qu'une seule tâche bien définie. Pour exécuter plusieurs tâches, il est parfaitement autorisé – et

Copyright © 2016 Eyrolles.

souhaitable – qu'une procédure en appelle une autre. Le fait d'imbriquer des procédures de cette manière vous offre une grande flexibilité dans la mise en place et l'organisation de vos programmes.

CarresPivotes.sb2

Pour le voir en action, commençons avec la procédure **Carré** que nous avons écrite dans la section précédente (figure 4-17). À présent, nous allons créer une nouvelle procédure, appelée **Carrés**, qui dessine quatre carrés étirés, comme sur la figure 4-19. Pour ce faire, **Carrés** appelle quatre fois **Carré**. Chaque appel utilise un argument différent, ce qui donne en sortie quatre carrés qui ont un coin en commun.

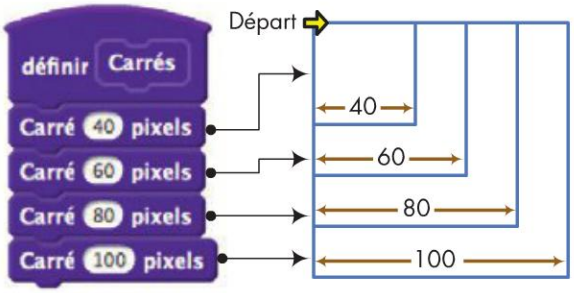


Figure 4-19 : La procédure **Carrés** *et sa sortie*

Nous pouvons à présent utiliser **Carrés** pour créer des dessins intéressants. La figure 4-20 montre une autre procédure, appelée **CarrésPivotés**, qui appelle plusieurs fois **Carrés**, en faisant pivoter les formes d'un certain angle à chaque appel.

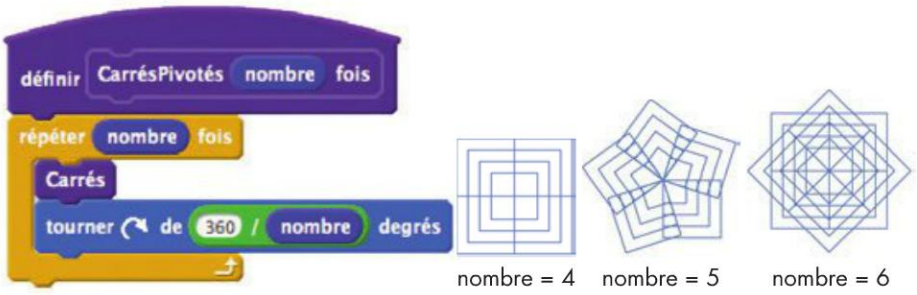


Figure 4-20 : La procédure **CarrésPivotés** *et quelques sorties possibles*

Dans cette procédure, le paramètre nombre est utilisé deux fois : une fois pour déterminer le nombre de répétitions et une autre fois pour calculer l'angle de rotation après l'appel de **Carrés**. Régler nombre à 5, par exemple, permettra de répéter le motif du carré de la figure 4-20 cinq fois avec une rotation vers la droite de 72° (soit 360° / 5) après chaque appel. Testez plusieurs valeurs pour découvrir de nouveaux motifs.

Dames.sb2

Voyons un autre exemple qui démontre la puissance des procédures imbriquées : nous commencerons avec la procédure **Carré** de la figure 4-16 pour finir avec un damier.

Créez la procédure **Ligne** qui dessine une ligne de carrés (figure 4-21). Notez que le nombre de carrés est spécifié dans un paramètre. Pour simplifier les choses, nous avons décidé de fixer la taille d'un carré à 20 pixels plutôt que de la définir dans un second paramètre de la procédure **Ligne**.

Copyright © 2016 Eyrolles.

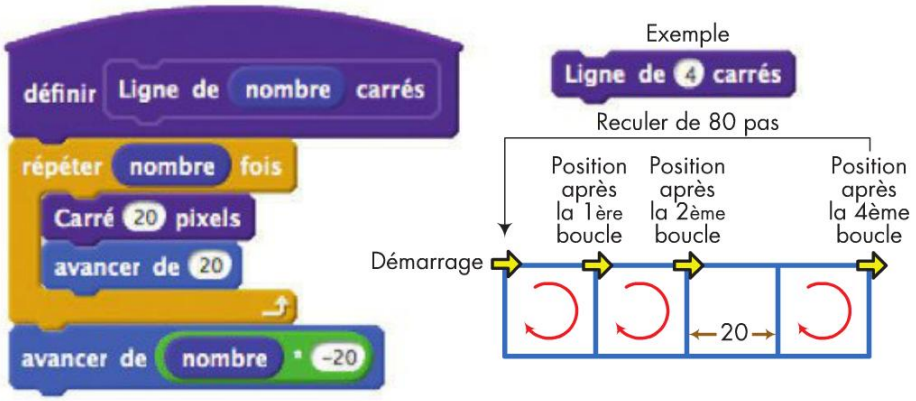


Figure 4-21 : La procédure **Ligne**

La figure 4-21 illustre également le résultat d'un appel de **Ligne** avec un argument de 4, qui engendre quatre appels de la procédure **Carré 20 pixels** dans la boucle. La position du lutin est ajustée après chaque dessin de carré pour régler la position initiale du carré suivant. Après avoir dessiné quatre carrés, la dernière commande renvoie le lutin à sa position initiale.

Pour dessiner une autre ligne de carrés sous la première, nous n'avons plus qu'à déplacer le lutin de 20 pixels vers le bas et à rappeler la procédure **Ligne**. Cette opération peut être répétée pour dessiner autant de lignes que nous le souhaitons. C'est ce qu'effectue notre procédure **Damier** (figure 4-22).

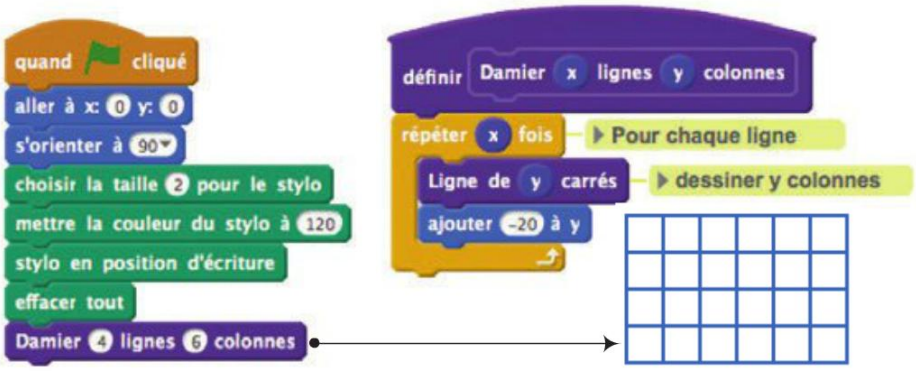


Figure 4-22 : La procédure **Damier** *et sa sortie*

Cette procédure prend deux paramètres en entrée : le nombre de lignes et le nombre de colonnes pour le damier désiré. Après avoir dessiné chaque ligne, la procédure déplace le lutin de 20 pixels vers le bas pour se préparer à dessiner la prochaine.

Les exemples présentés dans cette section montrent comment les procédures peuvent aider à diviser un programme en parties plus petites et plus simples à gérer. Une fois que vos procédures sont écrites et testées, elles peuvent être utilisées comme blocs de construction pour d'autres procédures plus complexes, sans avoir à se soucier des détails des implémentations de bas niveau. Vous pouvez alors vous concentrer sur la tâche importante, qui consiste à mettre en forme l'application complète en utilisant ces procédures comme blocs de construction.

Copyright © 2016 Eyrolles.

FAITES LE TEST 4-2

Que pensez-vous qu'il se passera si vous réglez la position initiale à 0° (vers le haut) au lieu de 90° (vers la droite) ? Le script fonctionnera-t-il ? Si non, comment pourriez-vous le réparer ? Apportez les changements nécessaires et exécutez le script pour vérifier votre réponse.

Travailler avec les procédures

Maintenant que vous avez saisi l'importance de diviser votre programme en petites parties à aborder une par une, voyons comment effectuer cette division. Chaque problème est différent et il n'existe pas de solution universelle – mais c'est ce qui rend ce jeu de réflexion amusant !

Dans cette section, nous étudierons d'abord le *processus descendant*, qui consiste à diviser de gros programmes en parties modulaires avec une structure claire et logique. Nous verrons ensuite une autre manière de construire des programmes complexes : le *processus montant*, qui consiste à combiner des procédures existantes. La figure 4-23 offre une vision globale de ces deux approches.

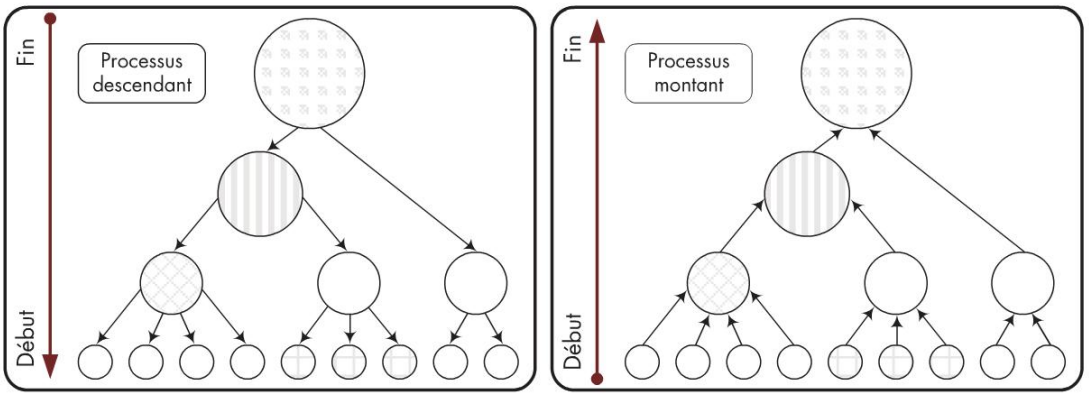


Figure 4-23 : Illustration des approches descendante (à gauche) et montante (à droite)

Dans chaque diagramme, le problème que nous voulons résoudre se trouve en haut et les étapes individuelles qui permettent de construire notre solution se trouvent en bas. Vous pouvez commencer du niveau qui vous semblera le mieux adapté.

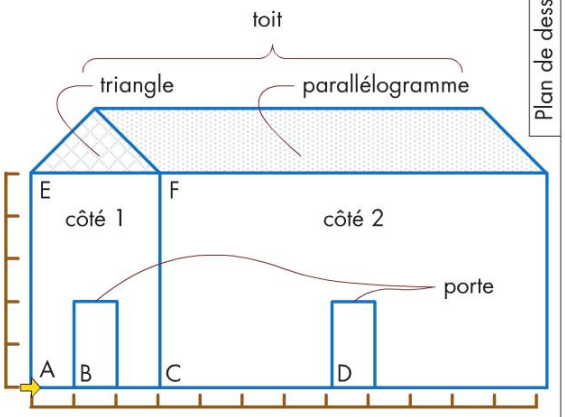
Processus descendant : diviser des programmes en procédures

La première étape dans la résolution d'un problème de programmation est la compréhension complète du problème. Après cela, vous pourrez planifier une solution générale et la diviser en tâches majeures. Il n'y a pas de solution juste ou fausse pour ce qui est de la division d'un programme et, avec l'expérience, vous améliorerez votre conception de ce qu'est une tâche majeure. Le fait de travailler à partir d'une solution générale en descendant vers des choses plus spécifiques vous assure au moins une logique générale correcte du programme.

Copyright © 2016 Eyrolles.

Maison.sb2

Pour vous montrer cette stratégie de résolution de problème, voyons comment nous dessinerions une maison similaire à celle représentée sur la figure 4-24.



Plan de dessin

1. Dessiner le côté 1. Le lutin finira sur le point A en pointant vers la droite.
2. Se déplacer horizontalement de 1 unité et dessiner la première porte. Le lutin finira au point B en pointant vers la droite.
3. Se déplacer horizontalement de 2 unités (jusqu'au point C) et dessiner le côté 2. Le lutin finira sur le point C en pointant vers la droite.
4. Se déplacer horizontalement de 4 unités (jusqu'au point D) et dessiner la seconde porte. Le lutin finira au point D et pointera vers la droite.
5. Se déplacer en arrière de 7 unités puis vers le haut de 5 unités. Ceci place le lutin sur le point E, pointant vers la droite.
6. Dessiner le toit. La procédure tracera d'abord le triangle, déplacera le lutin au point F, puis dessinera le parallélogramme.

Figure 4-24 : Nous pouvons dessiner cette maison en divisant le travail en plusieurs tâches plus petites et en les gérant une par une.

D'un certain côté, le fait de travailler sur ce problème simple nous permet de nous concentrer sur la stratégie de résolution en évitant de nous enliser dans de trop nombreux détails. De l'autre, malgré son apparente simplicité, ce problème connaît en réalité de nombreuses solutions différentes. En voici quelques-unes.

- Nous pouvons voir la maison comme constituée de lignes droites. Dans ce cas, le dessin de chaque ligne est une tâche majeure.
- Nous pouvons voir la maison comme constituée de six formes indépendantes : côté 1, côté 2, deux portes, un triangle et un parallélogramme. Dessiner chaque forme constitue alors une tâche majeure.
- Puisque les deux portes sont identiques, nous pouvons définir une tâche majeure qui consiste à dessiner une porte et y faire appel deux fois.
- Nous pouvons voir le triangle et le parallélogramme comme une seule unité : le toit. Dans ce cas, le dessin du toit est une tâche majeure.
- Nous pouvons voir le côté 1 et sa porte comme une seule unité, l'avant de la maison. Dans ce cas, le dessin de l'avant de la maison est une tâche majeure.

Il y a encore beaucoup d'autres possibilités, mais celles-ci sont suffisantes pour illustrer l'idée. Il s'agit de regrouper les tâches en petites parties compréhensibles, afin de vous concentrer sur une partie à la fois. Si vous trouvez des parties similaires, essayez de trouver une solution commune et de l'appliquer à toutes ces parties.

En gardant ceci à l'esprit, notre plan de dessin de la maison est également décrit sur la figure 4-24. Il considère que le lutin démarre au point A en pointant vers la droite. Nous n'avons qu'à créer un script qui corresponde aux étapes définies dans ce plan. Nous écrirons donc une procédure (nommée **Côté1**) pour dessiner le côté gauche de la maison comme l'indique

Copyright © 2016 Eyrolles.

l'étape ❶. Nous écrirons également trois procédures (nommées **Porte**, **Côté2** et **Toit**) pour dessiner les deux portes, le côté droit de la maison et le toit (comme l'indiquent les étapes ❷, ❸, ❹ et ❻) et nous connecterons toutes ces procédures avec les commandes de mouvement appropriées.

Notre procédure **Maison** est représentée à la figure 4-25, accompagnée des étapes de dessin qui correspondent à chaque appel de procédure. Elle prend un paramètre unique en entrée (appelé **échelle**), qui spécifie la longueur d'une unité pour dessiner la maison. Vous pouvez voir que la procédure **Porte** a été utilisée deux fois et que la procédure **Toit** est responsable du dessin complet du toit.

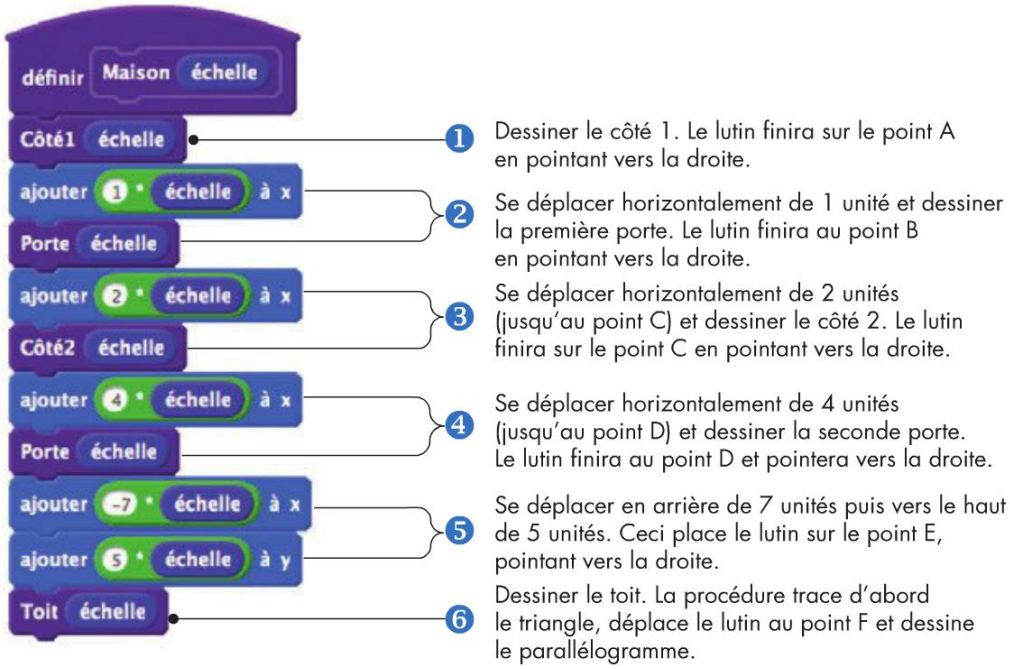


Figure 4-25 : La procédure **Maison**. *Les tâches majeures correspondent au plan de dessin.*

Les procédures individuelles servant à dessiner la maison sont représentées à la figure 4-26. Elles tracent des formes géométriques simples en utilisant des techniques que vous avez apprises au chapitre 2.

Les procédures **Côté1**, **Porte** et **Côté2** dessinent des rectangles de taille 3×5, 1×2 et 9×5 (redimensionnés par le facteur échelle), respectivement. La procédure **Toit** possède deux sous-procédures (nommées **Triangle** et **Parallélogramme**) qui dessinent les deux parties du toit. Notez que le facteur de redimensionnement échelle a été utilisé dans chacune de ces procédures.

FAITES LE TEST 4-3

Avez-vous remarqué que les procédures **Côté1**, **Porte** et **Côté2** utilisent des codes quasiment identiques ? Créez une nouvelle procédure appelée **Rectangle** qui prendra comme paramètres d'entrée la longueur, la largeur et l'échelle, puis qui dessinera un rectangle dans les dimensions spécifiées. Modifiez les procédures **Côté1**, **Porte** et **Côté2** pour y appeler la nouvelle procédure **Rectangle**.

Copyright © 2016 Eyrolles.



Figure 4-26 : Les procédures pour le dessin de la maison de la figure 4-24

Processus montant : construire un programme à partir de procédures

FloconDeFleur .sb2

Se concentrer d'abord sur des détails plus petits est une autre manière de faire face à un problème important. Si vous résolvez les petites parties (ou trouvez des solutions préexistantes), vous pouvez assembler les résultats de façon à atteindre une solution globale.

Pour démontrer cette technique de résolution de problèmes, commençons par une procédure simple (appelée **Feuille**) qui dessine une feuille unique (figure 4-27). Elle contient une boucle **répéter** qui s'exécute deux fois pour dessiner les deux moitiés de la feuille. Chaque moitié est dessinée par une série de 15 petits segments avec un angle de 6° entre chacun d'entre eux. La méthode est similaire à celle du dessin de polygones que nous avons vue au chapitre 2.

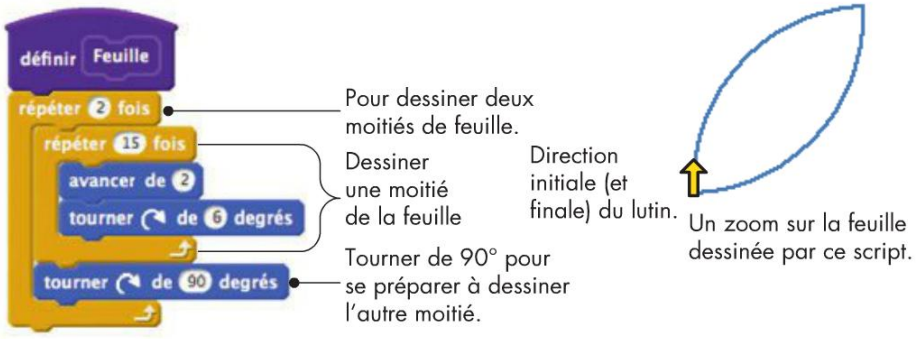


Figure 4-27 : La procédure **Feuille** *et sa sortie*

Copyright © 2016 Eyrolles.

En utilisant cette procédure présent dessiner une forme un peu plus complexe qui contient cinq feuilles. Notre nouvelle procédure **Feuilles** et sa sortie sont représentées à la figure 4-28. Comme vous pouvez le voir, nous n'avons eu qu'à appeler la procédure **Feuille** dans une boucle **répéter** avec un angle de rotation approprié entre chaque passage.

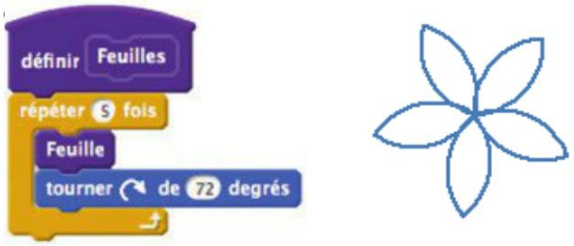


Figure 4-28 : La procédure **Feuilles** *appelle cinq fois* **Feuille** *avec un angle de rotation de 72° entre chaque appel.*

Nous pouvons à présent utiliser **Feuilles** et **Feuille** pour construire quelque chose d'encore plus complexe : une branche avec des feuilles. Notre procédure **Branche** et sa sortie sont illustrées à la figure 4-29. Le lutin avance de 40 pas, dessine une feuille unique (en appelant la procédure **Feuille**), avance de 50 pas de plus, dessine cinq feuilles (en appelant la procédure **Feuilles**) et retourne finalement à sa position initiale.

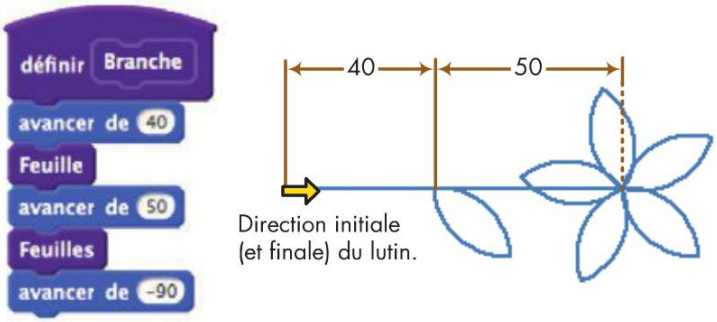


Figure 4-29 : La procédure **Branche** *et sa sortie*

Allons encore un pas plus loin en utilisant la procédure **Branche** pour le dessin de fleurs complexes. Notre nouvelle procédure **Fleur** et sa sortie sont représentées à la figure 4-30. Elle appelle simplement la procédure **Branche** six fois avec un angle de rotation de 60° entre chaque appel.

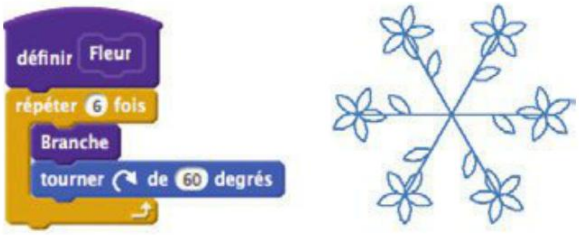


Figure 4-30 : La procédure **Fleur** *et sa sortie*

Nous pourrions encore continuer longtemps, mais le principe devrait à présent être clair. Nous avons commencé avec une procédure simple appelée **Feuille** et avons utilisé une nouvelle procédure (appelée **Feuilles**) pour créer un motif complexe. La procédure **Branche** s'est ensuite appuyée sur ces deux premières pour créer quelque chose de plus compliqué. La procédure **Fleur** a ensuite utilisé **Branche** pour dessiner un motif encore plus complexe. Si nous le voulions, nous pourrions créer une procédure qui dessinerait un arbre entier et une autre qui dessinerait un jardin rempli d'arbres.

Copyright © 2016 Eyrolles.

L'idée à retenir de cet exemple est la suivante : sans tenir compte de la complexité du problème que nous essayons de résoudre, nous pouvons toujours construire une solution en assemblant un certain nombre de parties plus petites et plus simples à gérer. Avec cette technique de résolution de problème, nous commençons avec de courtes procédures qui résolvent des problèmes très simples, puis nous les utilisons pour créer des procédures plus sophistiquées.

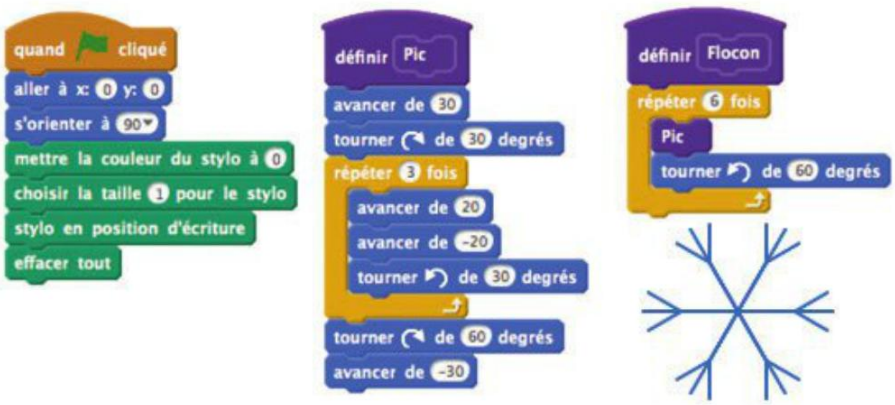
Résumé

Nous avons présenté dans ce chapitre une partie des concepts fondamentaux qui seront beaucoup utilisés dans le reste du livre. Tout d'abord, nous avons expliqué le concept d'envoi de messages pour la communication et la synchronisation entre lutins. Ensuite, nous avons présenté la programmation structurée et avons vu comment utiliser l'émission de messages pour implémenter ces procédures. Nous avons ensuite montré la fonction de création de blocs personnalisés de Scratch 2.0 et avons expliqué comment passer des arguments à des procédures pour rendre ces dernières plus flexibles. Nous avons vu plusieurs exemples qui expliquent comment diviser des problèmes importants en de petites parties plus simples à gérer et donc comment utiliser des procédures comme blocs de base pour créer des programmes importants. Enfin, nous avons étudié l'approche montante de résolution de problèmes, lors de laquelle nous avons assemblé des solutions simples pour résoudre un problème plus important.

Dans le chapitre suivant, vous apprendrez le concept le plus important de tout langage de programmation : les *variables*. Cette introduction aux variables sera le prochain pas essentiel pour devenir un programmeur compétent.

Problèmes

1. Écrivez différentes procédures pour dessiner chaque lettre de votre nom. Nommez chaque procédure avec la lettre qu'elle dessine. Écrivez ensuite un script qui les appelle afin de dessiner votre nom sur la scène.
2. Créez le programme présenté ci-après, exécutez-le et expliquez comment il fonctionne.

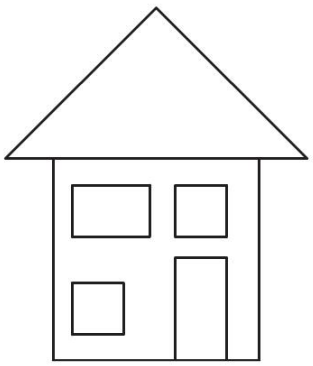


Copyright © 2016 Eyrolles.

3. Écrivez la procédure qui convertit les degrés Celsius en degrés Fahrenheit, comme présenté ci-après. Faites en sorte que le script arrondisse la réponse à l'entier le plus proche. Testez votre procédure avec différentes températures. Indice : °F = (9 / 5) × °C + 32.



4. Écrivez une procédure qui crée la maison de la figure. Commencez par écrire des procédures qui dessineront des petites parties de la maison (porte, toit, fenêtres). Combinez-les ensuite pour créer la maison entière.

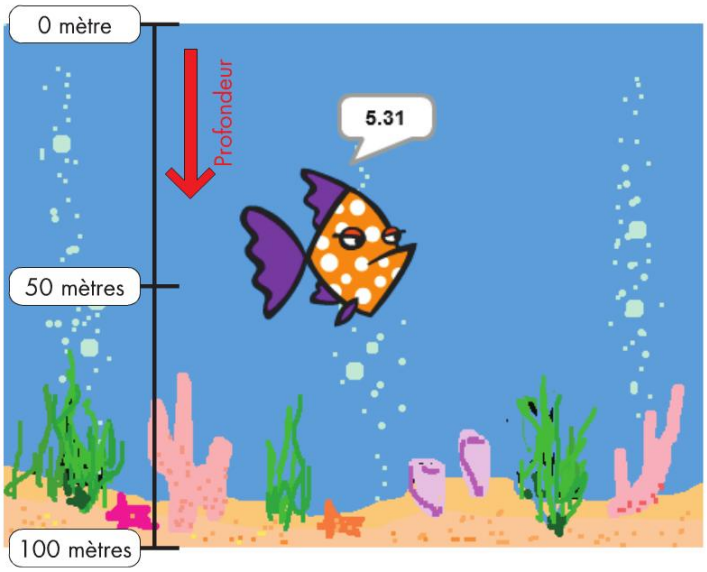


5. Écrivez une procédure qui calcule l'aire d'un cercle ($A = \pi r^2$) selon le rayon, comme présenté ci-après. Utilisez $\pi = 3{,}14$.



PressionSouslEau_PasDeSolution.sb2

6. Dans cet exercice, vous allez simuler l'expérience de la pression par un poisson sous l'eau. Soit la pression P (dans l'atmosphère) ressentie par un poisson, liée à sa profondeur p (en mètres sous la surface) par la relation $P = 0{,}1p + 1$. Le fichier *PressionSouslEau_PasDeSolution.sb2* contient une implémentation partielle de cette simulation. Terminez ce script pour que le poisson dise la pression qu'il ressent quand il nage, comme présenté ci-après.



Copyright © 2016 Eyrolles.

5

LES VARIABLES

Ce chapitre explique comment créer des scripts qui peuvent lire et enregistrer des valeurs. Avec des variables, vos applications interagissent avec les utilisateurs et répondent à leurs actions. Voici ce que nous allons découvrir dans ce chapitre :

- quels sont les types de données supportés par Scratch ;
- comment créer et manipuler des variables ;
- comment communiquer avec l'utilisateur pour écrire des programmes interactifs.

Les programmes que vous avez écrits jusqu'à présent vous ont aidé à acquérir d'importantes compétences en programmation Scratch. Toutefois, ils manquaient tout de même d'éléments pour la création d'applications à plus grande échelle. Les programmes plus complexes savent enregistrer des valeurs et décider d'effectuer une action sous certaines conditions. Ce chapitre prendra en charge le premier des deux aspects et la prise de décision sera discutée au chapitre suivant.

Copyright © 2016 Eyrolles.

Les scripts traitent et manipulent différents types de données lors de leur exécution. Ces données peuvent être des entrées de blocs de commandes (par exemple, le nombre 10 dans la commande **avancer de 10** et la chaîne de caractères « Hello ! » dans la commande **dire Hello !**), des sorties de blocs de commandes (comme **souris x**, **ordonnée y** et **nombre aléatoire**), ou des données entrées par l'utilisateur en réponse à une commande **demander et attendre**. Pour des programmes plus complexes, vous aurez souvent besoin d'enregistrer et de modifier des données pour accomplir certaines tâches. Dans Scratch, la gestion de données s'effectue grâce aux *variables* et aux *listes* (nous présenterons ces dernières au chapitre 9).

Ce chapitre commence par un récapitulatif des types de données que Scratch reconnaît. Il fournit ensuite une introduction aux variables et une discussion sur la façon de les créer et de les utiliser dans un programme. Nous verrons ensuite comment utiliser les moniteurs de variables dans plusieurs applications intéressantes. Après avoir compris ces concepts de base, vous apprendrez comment utiliser la commande **demander et attendre** pour obtenir des données entrées par l'utilisateur.

Les types de données dans Scratch

De nombreux programmes informatiques manipulent différentes sortes de données, dont les nombres, les textes, les images, etc. C'est une tâche de programmation importante. Vous devrez donc connaître les types de données et les opérations pris en charge par Scratch.

Pour ce qui est des types de données, Scratch propose les nombres, les chaînes de caractères et les booléens.

Une variable *booléenne* ne peut prendre que deux valeurs : vrai ou faux. Vous pouvez utiliser ce type de données pour tester une condition et faire choisir à votre programme un chemin d'exécution différent selon le résultat. Nous étudierons les booléens en détail dans le chapitre suivant.

Une variable *numérique* peut contenir une valeur entière ou une valeur décimale. Scratch ne fait aucune distinction : les deux types sont classés comme des « nombres ». Vous arrondissez des nombres décimaux vers l'entier le plus proche en utilisant le bloc **arrondir** de la palette *Opérateurs*. Vous pouvez également utiliser les fonctions **plancher de** ou **plafond de**, disponibles à partir du menu déroulant du bloc **racine de** (de cette même palette) pour obtenir respectivement l'entier précédant ou suivant un nombre décimal. Par exemple, **plafond de 3.9** donne 4 et **plancher de 3.9** donne 3.

Une *chaîne de caractères* contient des lettres (majuscules et minuscules), des chiffres et tout autre symbole que vous pourrez trouver sur votre clavier d'ordinateur (+, –, &, @, etc.).

Copyright © 2016 Eyrolles.

Quelle forme choisir ?

Avez-vous remarqué que les blocs de Scratch et leurs cases de paramètres ont des formes géométriques particulières ? Par exemple, la case de paramètre du bloc **avancer de 10** est un rectangle aux coins arrondis. La forme

correspond au type de donnée que la case de paramètre accepte. Essayez d'entrer votre nom (ou un autre texte) dans le bloc **avancer de 10** ; vous verrez que Scratch ne vous laissera entrer que des chiffres dans des cases rectangulaires-arrondies.

De même, la forme d'un bloc de fonction indique le type de données qu'il renvoie. La signification des différentes formes est présentée à la figure 5-1.

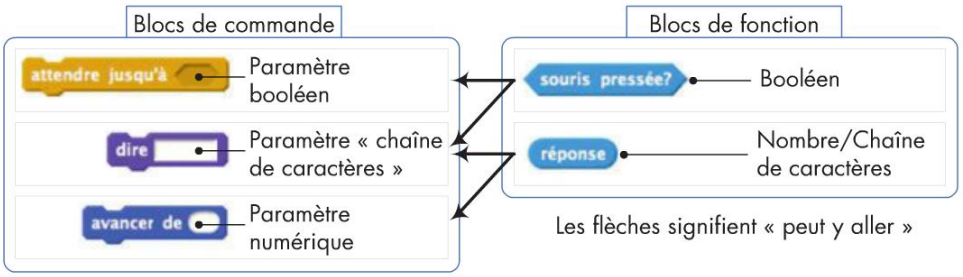


Figure 5-1 : La signification des formes des blocs de commande et de fonction

Les cases de paramètres peuvent prendre trois formes (hexagone, rectangle et rectangle-arrondi). Chacune est associée à un type de données particulier. Cependant, vous pouvez remarquer qu'un bloc de fonction rectangulaire arrondi peut renvoyer aussi bien un nombre qu'une chaîne de caractères.

Les cases hexagonales et rectangulaires-arrondies ne prennent que les blocs de fonction qui ont la même forme, alors qu'une case rectangulaire acceptera n'importe quel bloc de fonction. La bonne nouvelle est que Scratch vous empêche de faire des erreurs de types, donc vous n'avez pas besoin de retenir ces règles. Essayez de faire glisser un bloc hexagonal dans une case rectangulaire-arrondie ; cela ne fonctionnera pas car leurs types sont incompatibles.

Conversion automatique de types de données

Une case de paramètre numérique n'accepte que des blocs de fonction rectangulaires-arrondis. Tous les blocs de fonction rectangulaires-arrondis utilisés jusqu'à présent – dont **abscisse x**, **ordonnée y**, **direction**, **costume n°**, **taille**, **volume**, **tempo**, etc. – renvoient des nombres. Il est donc possible de les utiliser à l'intérieur d'une case numérique (telle que celle du bloc **avancer de 10**). Cependant, certains, comme le bloc **répondre** de la palette *Capteurs* ou encore le bloc **regroupe** de la palette *Opérateurs*, peuvent contenir un nombre aussi bien qu'une chaîne de caractères. Ceci soulève une question importante : que se passera-t-il si, par exemple, nous insérons un bloc contenant une chaîne de caractères dans une case numérique ? Heureusement, Scratch essaie de convertir automatiquement les types de données de la bonne manière.

Dans l'exemple de la figure 5-2, l'utilisateur entre `125` en réponse au message Entrez un nombre. Cette saisie est enregistrée dans le bloc de fonction **réponse**. Lorsque ce bloc est passé à la commande **dire**, le nombre est automatiquement converti en chaîne de caractères. Lorsque **réponse** est passé au bloc d'addition (qui s'attend à recevoir un nombre), il est converti en nombre, ici 125. Lorsque l'opération est effectuée, le résultat (25 + 125 = 150) est converti en chaîne de caractères et « 150 » est envoyé au bloc **dire**. Scratch tente automatiquement de gérer ces conversions pour vous.

Copyright © 2016 Eyrolles.

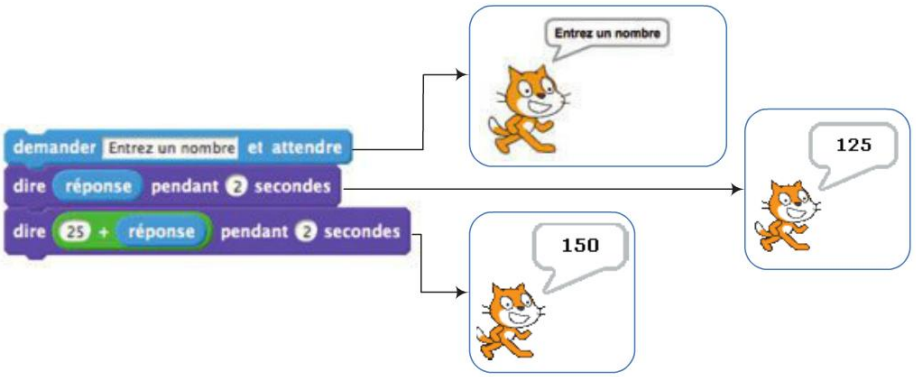


Figure 5-2 : Scratch convertit automatiquement les données selon le contexte.

Comprendre les types de données disponibles dans Scratch, ainsi que les opérations permises pour chaque type et la façon dont Scratch les convertit, vous sera d'une grande aide pour comprendre la façon dont les choses fonctionnent. Dans la section suivante, vous étudierez les variables et vous verrez comment les utiliser pour enregistrer et manipuler des données dans vos programmes.

Introduction aux variables

Imaginons que nous voulions créer une version logicielle du fameux jeu de Tape-taupe. Le jeu original possède une surface plate avec un certain nombre de trous. Le joueur utilise un maillet pour taper les taupes lorsqu'elles sortent de l'un de ces trous. Dans notre version, un lutin apparaît à des lieux aléatoires sur la scène, reste visible pendant un court instant, puis disparaît pou réapparaître à un autre endroit. Le joueur doit cliquer sur le lutin dès qu'il le voit ; à chaque fois qu'il y parvient, il gagne un point. La question à se poser en tant que programmeur est la suivante : comment garder une trace du score du joueur ? Bienvenue dans le monde des variables !

Dans cette section, je présenterai les variables, l'un des éléments les plus importants de tout langage de programmation. Vous apprendrez comment créer des variables dans Scratch et comment les utiliser pour enregistrer différents types de données. Vous étudierez également les blocs qui permettront d'en régler et d'en changer les valeurs dans vos programmes.

Qu'est-ce qu'une variable ?

Une *variable* est un emplacement dans la mémoire de votre ordinateur qui porte un nom. Voyez-la comme une boîte qui stocke des données comme des nombres ou du texte, à laquelle un programme pourra avoir accès lorsqu'il en aura besoin. La figure 5-3, par exemple, représente une variable nommée côté dont la valeur actuelle est 50.

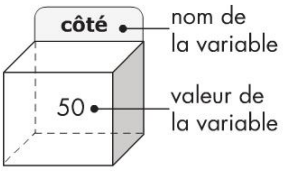


Figure 5-3 : Une variable est comme une boîte qui porte un nom et qui contient une valeur.

Lorsque vous créez une variable, votre programme réserve assez de mémoire pour stocker sa valeur et donne le nom de la variable à cet

Copyright © 2016 Eyrolles.

emplacement mémoire. Ensuite, vous utilisez son nom dans le programme pour obtenir la valeur qu'elle représente. Par exemple, si nous avons une boîte (une variable) nommée côté qui contient le nombre 50, nous pouvons construire une commande telle que **avancer de (3*côté)**. Quand Scratch exécutera cette commande, il localisera la boîte nommée côté dans la mémoire de l'ordinateur, extraira son contenu (dans notre cas, le nombre 50) et utilisera sa valeur pour remplacer **côté** dans le bloc **avancer de (3*côté)**. Le lutin avancera donc de 150 pas (3 × 50 pas).

Dans notre jeu de Tape-taupe, nous devons nous souvenir du score du joueur. Pour ce faire, réservons un emplacement mémoire (comme une boîte) dans notre ordinateur pour stocker le score. Nous devons lui donner un nom unique, disons score, qui nous permettra de la retrouver dès que nous voudrons connaître ou changer ce qui s'y trouve.

Au démarrage de notre jeu, nous dirons à Scratch de « mettre score à 0 » ; il ira chercher la boîte nommée score et y stockera la valeur 0. Nous lui dirons également « d'ajouter 1 à score » à chaque fois que le joueur aura réussi à cliquer sur une taupe. En réponse au premier clic réussi, Scratch regardera à nouveau à l'intérieur de la boîte nommée score, trouvera notre 0, y ajoutera 1 et rangera le résultat, 1, dans la boîte. Au prochain clic réussi, Scratch suivra de nouveau notre commande « ajouter 1 à score » et stockera le résultat, 2, dans la boîte. Pour l'instant, souvenez-vous que la valeur de score change durant l'exécution du programme. Voilà pourquoi elle est appelée *variable* : sa valeur varie.

Une utilisation importante des variables est le stockage des résultats intermédiaires d'une expression algébrique, de la même manière que nous effectuons nos calculs mentaux. Par exemple, si vous deviez trouver le résultat de 2 + 4 + 5 + 7, pour commenceriez sans doute par ajouter 2 à 4 et retiendriez la réponse (6), à laquelle vous ajouteriez ensuite 5 et mémoriseriez la nouvelle réponse, 11. Enfin, vous ajouteriez 7 au résultat précédent pour obtenir la réponse finale de 18.

Pour illustrer la façon dont les variables peuvent être utilisées pour un stockage temporaire, imaginons que vous vouliez écrire un code qui calcule l'expression suivante :

$$\frac{(1/5)+(5/7)}{(7/8)-(2/3)}$$

Vous pourriez calculer l'opération complète en une seule commande, mais cela rendrait la commande bien difficile à lire, comme le montre l'image ci-dessous :



Une autre manière d'écrire ce programme serait de calculer le numérateur et le dénominateur séparément et d'utiliser ensuite un bloc **dire** pour afficher le résultat de leur division. Nous pouvons faire ceci en créant deux variables nommé num (pour numérateur) et den (pour dénominateur) et en réglant leurs valeurs comme sur la figure 5-4.

Copyright © 2016 Eyrolles.

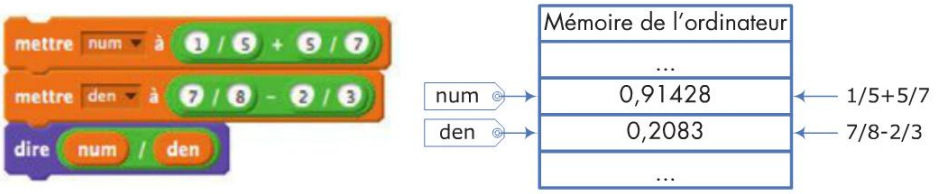


Figure 5-4 : Deux variables num *et* den *retiennent les valeurs, respectivement du numérateur et du dénominateur de notre expression.*

Regardez la façon dont sont organisées les variables dans la mémoire de l'ordinateur. Ici, num est comme une étiquette attachée à l'emplacement mémoire où le résultat de l'opération (1/5 + 5/7) est stocké. De même, den est attaché à l'emplacement où (7/8 – 2/3) est stocké. Lorsque la commande **dire** est exécutée, Scratch va chercher le contenu des emplacements mémoire nommés num et den, divise les deux nombres correspondants et envoie le résultat à la commande **dire** pour l'afficher.

Nous pourrions encore diviser cette opération en calculant chaque fraction individuellement avant d'afficher le résultat de l'opération complète (figure 5-5).

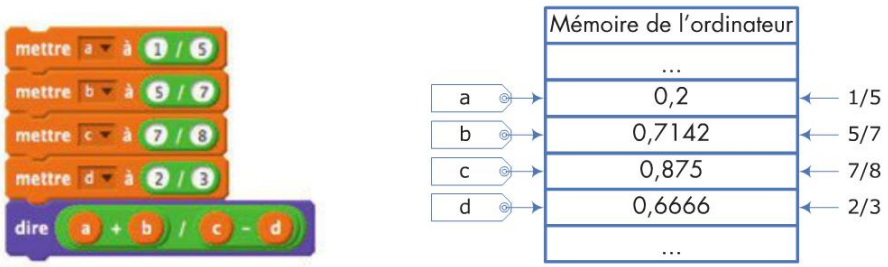


*Figure 5-5 : Utilisation de quatre variables (*a, b, c *et* d*) pour stocker les quatre fractions de l'expression.*

Ici, nous utilisons quatre variables (nommées a, b, c et d) pour stocker les quatre fractions de notre expression mathématique. La figure représente également l'allocation mémoire.

Bien que ces trois programmes donnent la même réponse, chaque implémentation suit un style différent. La première met tout dans une seule expression, ce qui est compliqué à lire. La troisième divise les choses à un niveau très détaillé, mais cela peut être également difficile à lire. La seconde solution divise l'expression à un niveau raisonnable et utilise des variables qui non seulement rendent le programme simple à comprendre mais montrent également les parties principales de l'expression (le numérateur et le dénominateur). Cette solution est parfaite.

Cet exemple simple démontre bien le fait qu'un problème puisse avoir plusieurs solutions. Vous pourrez parfois vous soucier plutôt de la vitesse d'exécution et de la taille d'un programme, parfois plutôt de sa lisibilité. Puisque vous lisez un livre d'introduction à la programmation, les scripts qui y sont présents se soucient plutôt de la lisibilité.

Maintenant que vous comprenez ce que sont les variables et pourquoi les utiliser, allons un peu plus loin dans nos applications Scratch en créant des variables.

Copyright © 2016 Eyrolles.

Créer et utiliser des variables

SimulateurDe Dés_PasDeCode .sb2

Dans cette section, nous verrons comment créer et utiliser des variables grâce à une application simple qui simule le lancer d'une paire de dés et affiche leur somme (figure 5-6).

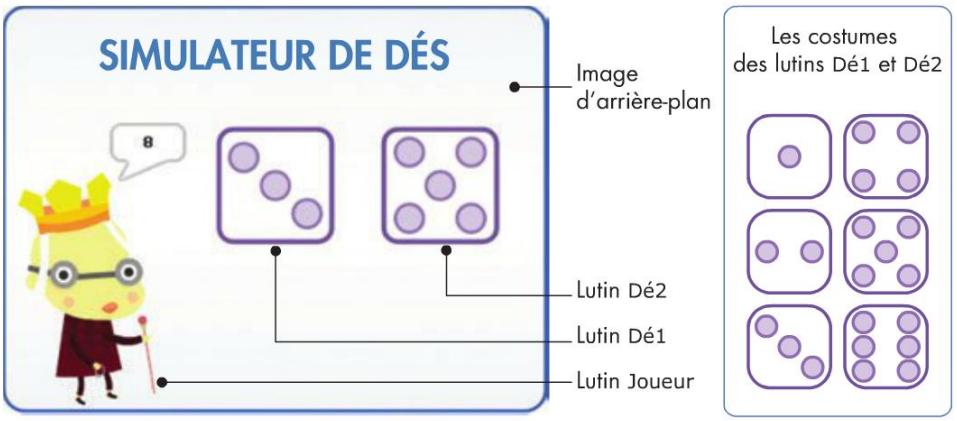


Figure 5-6 : L'interface utilisateur du simulateur de dés

Notre simulateur contient trois lutins : Joueur, Dé1 et Dé2. Le lutin Joueur gère la simulation. Lorsqu'on clique sur le drapeau vert, il génère deux nombres aléatoires compris entre 1 et 6, qu'il sauvegarde dans deux variables nommées aléa1 et aléa2. Il envoie ensuite un message aux deux autres lutins, Dé1 et Dé2, pour qu'ils affichent respectivement aléa1 et aléa2. Ensuite, le lutin Joueur ajoute aléa1 et aléa2 et affiche leur somme en utilisant le bloc **dire**.

Construisons cette application avec l'approche montante. Ouvrez le fichier *SimulateurDeDés_PasDeCode.sb2*. Il contient l'image d'arrière-plan pour la scène, ainsi que les trois lutins. Nous allons créer un par un les scripts dont nous avons besoin.

Tout d'abord, cliquez sur la miniature du lutin Joueur pour le sélectionner. Allez dans la palette *Données* et cliquez sur **Créer une variable** (figure 5-7 à gauche). Dans la fenêtre qui apparaît (à droite de la figure), entrez le nom de la variable et sélectionnez sa portée. La *portée* d'une variable détermine quels lutins peuvent en écrire ou en changer la valeur. Pour cet exemple, entrez aléa1 comme nom de variable et sélectionnez l'option **Pour tous les lutins** pour la portée. Ceci fait, cliquez sur **OK**.

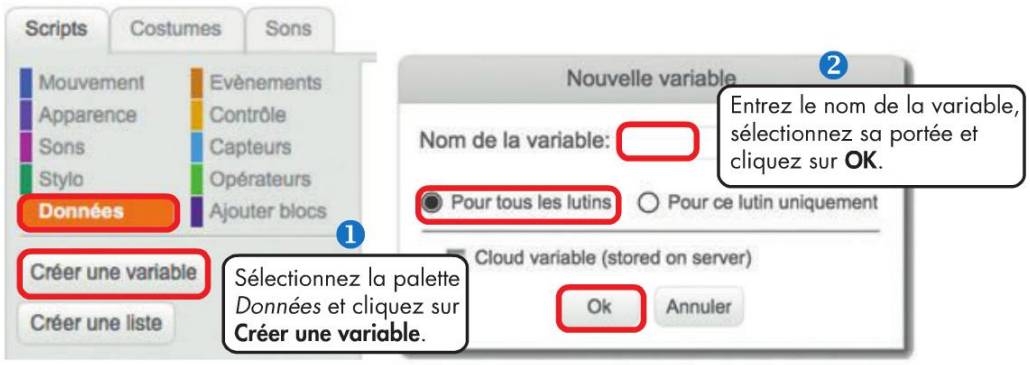


Figure 5-7 : Créer une variable, la nommer et spécifier sa portée.

Copyright © 2016 Eyrolles.

Plusieurs nouveaux blocs apparaissent dans la palette *Données* (figure 5-8).

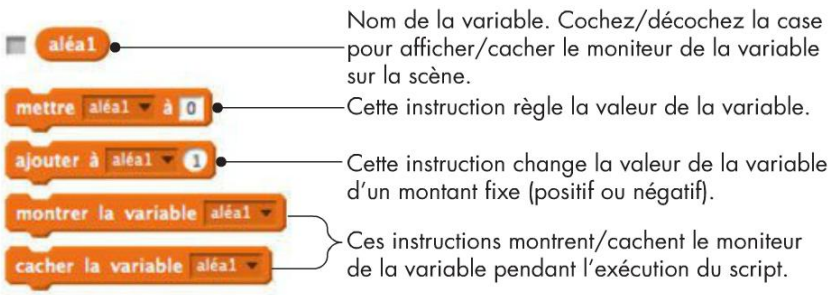


Figure 5-8 : Les nouveaux blocs qui apparaissent lorsque vous créez la variable aléa1

Vous pouvez utiliser ces blocs pour régler une variable à une valeur spécifique, changer sa valeur d'un montant fixe et montrer ou cacher son moniteur sur la scène. Le *moniteur* d'une variable, comme vous le verrez plus loin, en affiche la valeur actuelle.

NOMMER LES VARIABLES

Au fil des années, de nombreuses manières de nommer les variables sont apparues. Une convention populaire consiste à démarrer le nom par une lettre minuscule et à mettre en majuscule la première lettre de tout mot additionnel, tel que longueurDeCôté, nomDeFamille, ou valeurAbsolue.

Bien que Scratch accepte des noms de variable qui commencent par des nombres et qui contiennent des espaces (par exemple, 123côté ou longueur de côté), ce n'est pas le cas de la plupart des langages de programmation. Je vous recommande donc de ne pas utiliser ces noms inhabituels pour vos variables.

Et, bien que vous puissiez donner le nom de votre choix aux variables, je vous recommande d'utiliser des noms descriptifs qui aient du sens. Les variables dont le nom est une lettre unique telle que w ou z devraient être exclues, à moins que leur sens soit très clair. D'un autre côté, les noms trop longs rendent parfois votre script plus difficile à lire.

De plus, notez que les noms de variables dans Scratch tiennent compte des majuscules, ce qui signifie que côté, CÔTÉ et cÔTÉ sont toutes des variables uniques. Pour éviter les confusions, faites en sorte de ne pas utiliser des variables dont le nom ne diffère que par les majuscules dans un même script.

Répétez la procédure pour créer une nouvelle variable, nommée aléa2. La palette *Données* devrait contenir un nouveau bloc de variable et la petite flèche noire des blocs de la figure 5-8 devrait vous permettre de choisir entre aléa1 et aléa2. Maintenant que ces deux variables sont créées, nous pouvons construire le script du lutin Joueur, présenté à la figure 5-9.



Figure 5-9 : Le script du lutin Joueur

Copyright © 2016 Eyrolles.

La première commande affecte à aléa1 un nombre aléatoire compris entre 1 et 6. Repensez à notre analogie de la boîte : cette commande indique à notre lutin de trouver la boîte nommée aléa1 et d'y ranger le nombre généré aléatoirement. La seconde commande affecte à aléa2 un deuxième nombre aléatoire compris entre 1 et 6. Ensuite, le script envoie un message nommé Lancer aux deux autres lutins (Dé1 et Dé2) pour leur indiquer de changer de costume en fonction de la valeur de aléa1 et aléa2. Une fois que les lutins Dé1 et Dé2 ont terminé leur tâche, le script reprend son exécution et affiche la somme des nombres affichés sur les dés grâce à un bloc **dire**. Jetons un œil au script de gestion du message Lancer du lutin Dé1 (figure 5-10).

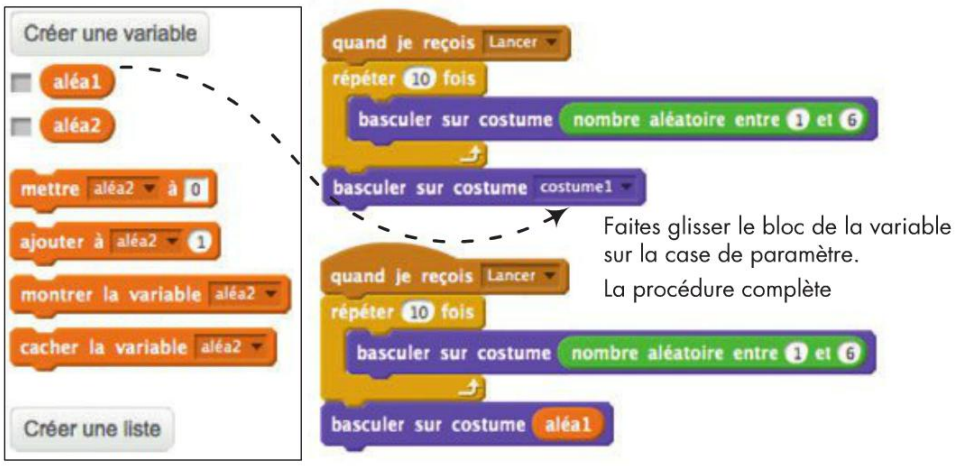


Figure 5-10 : Pour utiliser une variable dans un bloc de commande, faites-la simplement glisser sur la case de paramètre du bloc.

Après avoir créé le script présenté en haut à droite de la figure, faites glisser le bloc aléa1 de la palette *Données* vers la case de paramètre du bloc **basculer sur costume** pour créer le script complet (en bas à droite). Dans ce script, le bloc **répéter** change 20 fois le costume du dé aléatoirement pour simuler un lancer (vous pouvez changer ce nombre si vous le désirez). Ensuite, le dé mettra le costume spécifié par la variable aléa1. Pour rappel, chaque dé possède six costumes correspondant aux nombres 1 à 6, dans l'ordre. Cela signifie que si aléa1 contenait 5, alors la dernière commande **basculer vers le costume** afficherait le costume contenant cinq points.

À présent, nous pouvons créer le script du lutin Dé2, qui sera quasiment identique à celui de Dé1. Tout ce qu'il vous reste à faire est de dupliquer le script de Dé1 et de remplacer aléa1 par aléa2. Notre simulateur de dé est à présent complet, alors testons-le. Cliquez sur le drapeau vert pour voir la simulation en action. Si l'application ne fonctionne pas, vérifiez dans le fichier *SimulateurDeDés.sb2* qui contient l'implémentation correcte du programme.

Copyright © 2016 Eyrolles.

FAITES LE TEST 5-1

Sélectionnez le lutin Joueur et créez une variable appelée somme. Réglez la portée de la variable sur Pour ce lutin uniquement. Modifiez le dernier bloc du script de Joueur pour utiliser cette nouvelle variable :



À présent, sélectionnez le lutin Dé1 (ou Dé2) et regardez sa palette *Données*. Sauriez-vous expliquer la raison pour laquelle vous n'y voyez pas la variable somme ?

La portée des variables

Un autre concept important lié aux variables est leur *portée*. La portée d'une variable détermine quel lutin peut en écrire ou en changer la valeur.

PortéeDémo.sb2

Vous spécifiez la portée d'une variable lorsque vous la créez, en sélectionnant l'une des deux options de la figure 5-7. Choisir Pour ce lutin uniquement crée une variable qui ne pourra être changée que par le lutin à qui elle appartient. Les autres peuvent toujours lire et utiliser la valeur contenue dans cette variable, mais ils ne pourront pas la changer (figure 5-11).



Figure 5-11 : Seul le lutin Chat *peut écrire dans la variable* valeur.

Dans cette figure, le lutin Chat possède une variable, nommée valeur, qui a comme portée Pour ce lutin uniquement. Le lutin Pingouin peut lire cette valeur avec le bloc **abscisse x de Pingouin** de la palette *Capteurs*. Lorsque vous sélectionnez Chat comme second paramètre de ce bloc, vous pourrez en choisir un attribut dans le premier paramètre, dont l'une de ses variables.

Cependant, Scratch ne fournit pas de bloc qui permette au lutin Pingouin de modifier la variable valeur. Ainsi, il ne peut pas agir sur cette variable et créer des effets indésirables dans les scripts exécutés par le lutin Chat. C'est une bonne manière de faire que de régler la portée d'une variable à Pour ce lutin uniquement lorsqu'elle ne doit être mise à jour que par ce dernier.

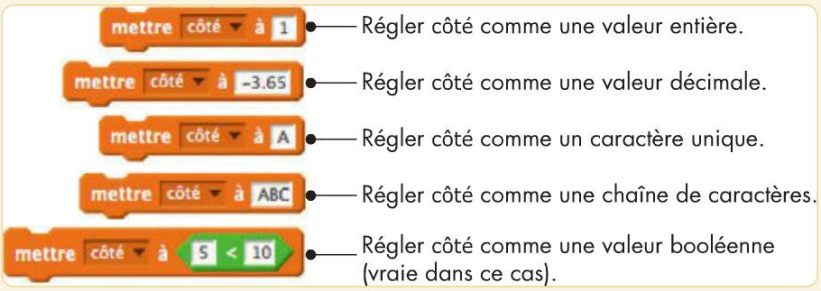
On dit des variables dont la portée a été réglée à Pour ce lutin uniquement qu'elles ont une *portée locale* et sont donc appelées *variables locales*. Différents lutins peuvent utiliser des noms identiques pour leurs variables locales sans créer de conflit. Par exemple, si vous avez deux lutins voiture dans un jeu de

Copyright © 2016 Eyrolles.

course, chacun d'entre eux peut avoir une variable locale nommée vitesse, qui caractérise le déplacement de la voiture sur la scène. Chaque lutin voiture peut changer sa variable vitesse indépendamment de l'autre. Cela signifie que, si vous réglez la variable vitesse de la première voiture à 10 et celle de la deuxième à 20, alors celle-ci ira plus vite que la première.

LE TYPE DE DONNÉE D'UNE VARIABLE

Vous vous demandez peut-être comment fait Scratch pour connaître le type de la donnée contenue dans une variable ? La réponse est la suivante : il n'en sait rien ! Lorsque vous créez une variable, Scratch n'a aucune idée de ce que vous comptez en faire. Il ne sait pas si ce sera un nombre, une chaîne de caractères ou un booléen. Toute variable peut contenir n'importe quel type de donnée. Par exemple, toutes les commandes suivantes sont valides dans Scratch.



Il ne tient qu'à vous de stocker les valeurs correctes dans vos variables. Cependant, n'oubliez pas que Scratch essaiera de convertir les types de données selon le contexte. Pour voir ce qui se passe lorsque vous stockez un type de donnée incorrect dans une variable, considérez les deux exemples suivants :



Puisque la commande **avancer de** s'attend à recevoir un paramètre numérique, Scratch convertit automatiquement la valeur stockée dans la variable côté en un nombre avant de la transmettre. Dans le premier script (à gauche), Scratch ne peut pas convertir la chaîne de caractères « Nonsense » en nombre. Plutôt que d'afficher un message d'erreur, Scratch règle silencieusement le résultat de la conversion à 0 et transmet cette valeur à la commande **avancer de**. Le lutin ne se déplace donc pas. D'un autre côté, dans le second script (à droite), Scratch ignore les espaces contenus dans la chaîne de caractères et transmet le résultat numérique au bloc **avancer de** ; le lutin avance de 100 pas. Notez que si le bloc cible s'attendait à recevoir une chaîne de caractères au lieu d'un nombre, Scratch aurait transmis la chaîne telle quelle, en incluant les espaces.

Copyright © 2016 Eyrolles.

Les variables dont la portée a été réglée à Pour tous les lutins, quant à elles, peuvent être lues et modifiées par n'importe quel lutin de votre application. Souvent appelées *variables globales*, elles sont utiles pour la communication et la synchronisation entre lutins. Par exemple, si un jeu contient trois boutons qui permettent à l'utilisateur de sélectionner un niveau dans lequel jouer, vous pouvez créer une variable globale nommée `niveauDuJeu` et faire en sorte que chaque bouton sache la régler différemment. Vous comprendrez ensuite facilement le choix de l'utilisateur en vous référant à cette variable.

Sélectionner l'option Pour tous les lutins permet également de modifier l'option *Cloud variable (stored on server)* (en français : *Variable du Cloud, stockée sur le serveur*, voir figure 5-7). Cette option n'est disponible que si vous avez le statut de « Scratcher » sur le site du logiciel. Lorsque vous ouvrez un compte, vous avez le statut de « Nouveau Scratcher ». Les conditions pour devenir un « Scratcher » sont les suivantes :

- avoir quelques projets partagés ;
- avoir été sur Scratch depuis au moins deux semaines ;
- être actif sur le site web.

Cette option permet de stocker une variable sur le serveur de Scratch, appelé *Cloud*. Les blocs de variables du Cloud sont accompagnés d'un nuage pour les distinguer des variables normales :



Quiconque ouvre un projet que vous avez partagé sur le site de Scratch peut en voir les variables du Cloud. Par exemple, si vous partagez un jeu, vous pouvez utiliser une variable de Cloud qui enregistrerait le meilleur score de tous les joueurs. La variable `score` devrait se mettre à jour presque immédiatement pour toute personne qui serait en train de jouer à votre jeu. Puisque ces variables sont stockées dans le serveur de Scratch, leurs valeurs sont sauvegardées même quand vous quittez votre navigateur. Les variables du Cloud rendent possible la création de sondages ou d'autres projets qui nécessitent la sauvegarde de données dans le temps.

Maintenant que vous savez ce qu'est la portée, il est temps d'apprendre à mettre à jour des variables – et d'utiliser ces connaissances pour créer des programmes plus intéressants.

Modifier les variables

Scratch fournit deux blocs de commande qui servent à modifier des variables. La commande **mettre à** affecte directement une nouvelle valeur à une variable, sans se soucier de ce qu'elle contenait précédemment. La commande **ajouter à**, quant à elle, est utilisée pour ajouter une valeur à celle qui était contenue dans la variable. Les trois scripts de la figure 5-12 montrent comment utiliser ces commandes de trois manières différentes pour obtenir le même résultat.

Copyright © 2016 Eyrolles.

Tous les scripts de la figure commencent par initialiser les variables somme et delta à 0 et 5 respectivement. Le premier script utilise la commande **ajouter à** pour incrémenter somme de la valeur de delta (somme = 0 + 5 = 5). Le second script utilise la commande **mettre à** pour enregistrer le résultat de l'addition de somme et de delta (0 + 5) dans la variable somme. Le troisième script donne le même résultat avec l'aide d'une variable temporaire nommée temp. Les valeurs de somme et de delta sont additionnées, le résultat est stocké dans temp et la valeur de temp est finalement copiée dans somme.

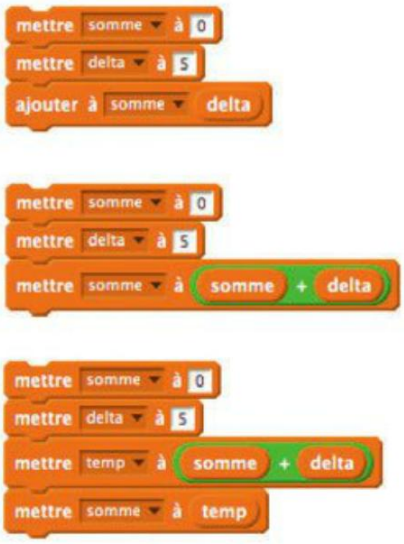


Figure 5-12 : Trois méthodes pour changer la valeur d'une variable

Après avoir exécuté l'un des scripts de la figure 5-12, somme contient le nombre 5 ; ces scripts sont donc équivalents du point de vue fonctionnel. Notez que la méthode utilisée dans le second est une pratique de programmation courante ; je vous recommande de l'étudier un moment pour vous sentir à l'aise avec elle. Voyons à présent la commande **ajouter à** en action.

Toile d'araignée

ToiledAraignée.sb2

Créons une toile d'araignée en dessinant plusieurs hexagones dont la taille augmente (figure 5-13). La procédure **Triangle** trace un triangle équilatéral avec une longueur de côté variable et **Hexagone** appelle six fois **Triangle** avec un angle de rotation de 60° (360°/6) après chaque appel. La figure montre comment l'hexagone est formé à partir de six triangles.

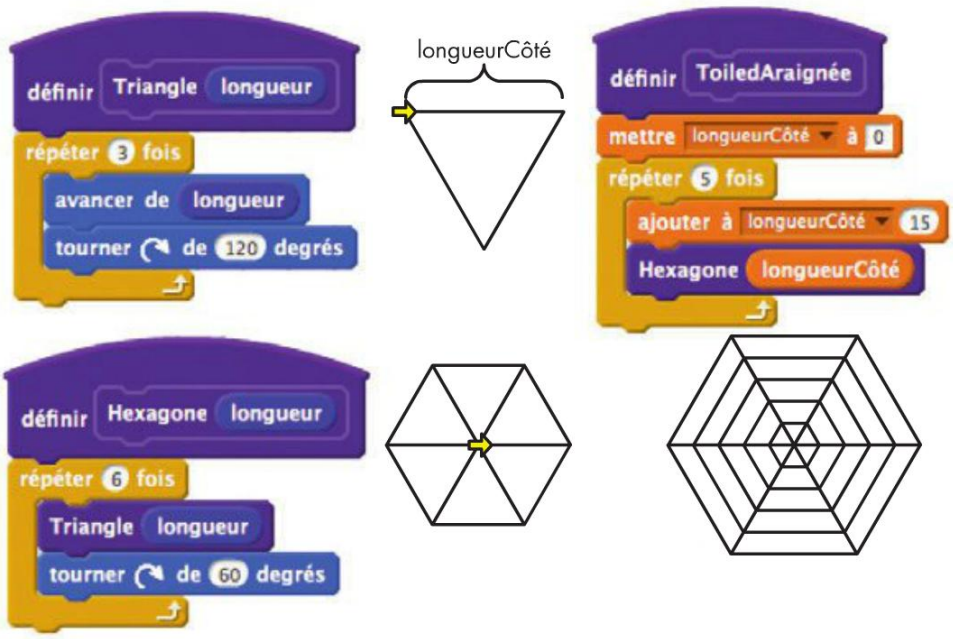


Figure 5-13 : Création d'une toile d'araignée en dessinant plusieurs hexagones de tailles croissantes

Copyright © 2016 Eyrolles.

La procédure **ToiledAraignée** appelle **Hexagone** à cinq reprises avec différentes valeurs pour longueurCôté à chaque répétition. Elle donne les hexagones concentriques de la figure. La commande **ajouter à** est utilisée pour régler la valeur de longueurCôté à l'intérieur de la boucle **répéter**. Reproduisez la procédure **ToiledAraignée**, exécutez-la et observez comment elle fonctionne.

Moulin

Moulin.sb2

Cet exemple ressemble au précédent, si ce n'est que nous utiliserons cette fois une variable pour contrôler le nombre de répétitions de triangles. La procédure correspondante (appelée **Pales**) est représentée à la figure 5-14. La procédure **Moulin**, présentée dans la même figure, fonctionne comme **ToiledAraignée**, sauf que nous avons changé la couleur du stylo à chaque passage dans la boucle pour donner un effet arc-en-ciel. Quelques résultats de la procédure **Moulin** sont présentés sur la figure. Expérimentez cette procédure avec différents nombres de pales pour voir ce que vous pouvez créer d'autre.

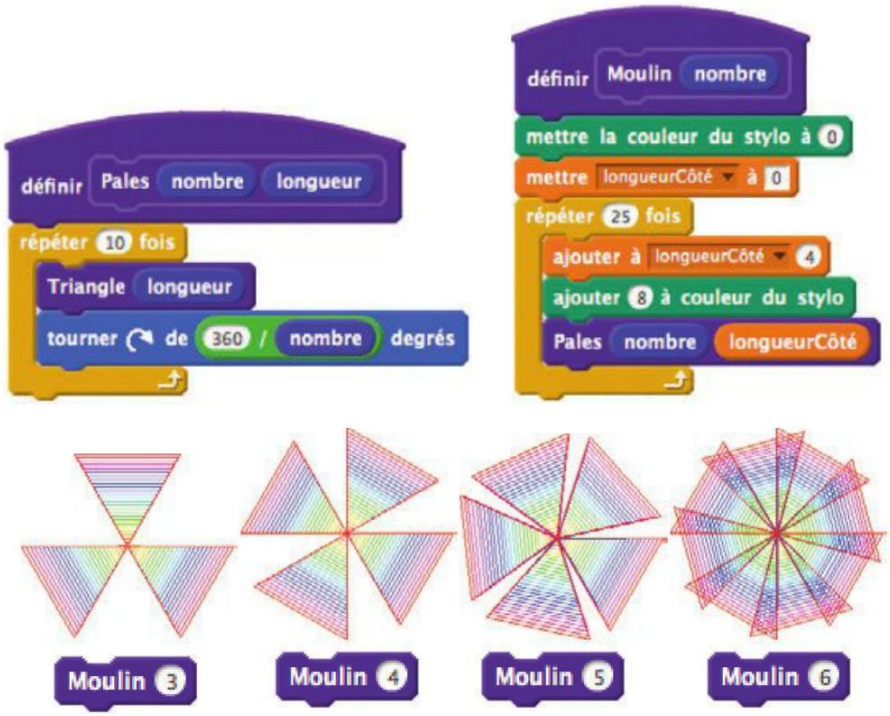


Figure 5-14 : Création d'un moulin par rotations multiples d'un triangle équilatéral

Maintenant que vous avez vu les principes fondamentaux des variables, vous vous demandez peut-être ce qui se passera lorsque vous dupliquerez un lutin. La copie partage-t-elle les variables du lutin parent, ou possède-t-elle ses propres variables locales ? Les clones ont-ils accès aux variables globales ? Nous répondrons à toutes ces questions dans la section suivante.

FAITES LE TEST 5-2

Modifiez le programme Moulin pour cacher le lutin. Le lutin ne masquant plus ses dessins, cela devrait simplifier votre observation des étapes de création.

Copyright © 2016 Eyrolles.

Les variables dans les clones

Chaque lutin possède sa propre liste de propriétés qui lui sont associées, dont ses positions *x* et *y*, sa direction, etc. Vous pouvez considérer cette liste comme un sac à dos contenant les valeurs actuelles des attributs du lutin (figure 5-15). Lorsque vous créez une variable locale, elle est ajoutée au sac à dos du lutin.

Lorsque vous clonez un lutin, le clone hérite de copies des attributs de son lutin parent, y compris ses variables. Une propriété héritée démarre identique à la propriété du parent au moment où le clone est créé. Mais après cela, si les attributs et les variables du clone changent, cela n'affectera pas ceux du parent. De même, les changements effectués sur le lutin parent n'affectent pas les clones.

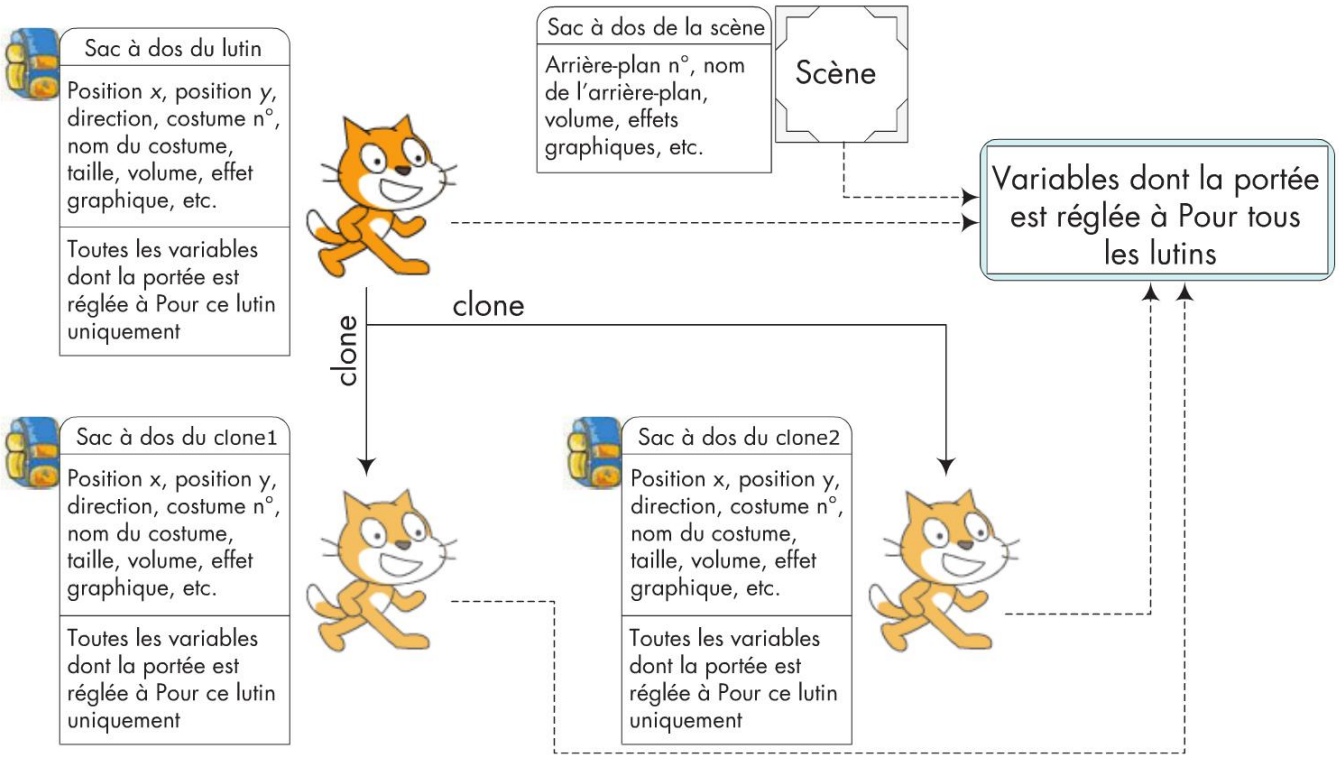


Figure 5-15 : Les clones héritent de copies des variables de leurs parents

Pour illustrer ce principe, disons que le lutin parent possède une variable nommée vitesse, dont la valeur actuelle est 10. Lorsque vous clonez le parent, le nouveau lutin aura aussi une variable nommée vitesse dont la valeur sera 10. Ensuite, si le parent change vitesse à 20, la valeur de vitesse du clone sera toujours 10.

CloneIDs.sb2

Vous pouvez utiliser ce concept pour différencier les clones les uns des autres dans votre application. Regardons par exemple le programme présenté à la figure 5-16.

Copyright © 2016 Eyrolles.

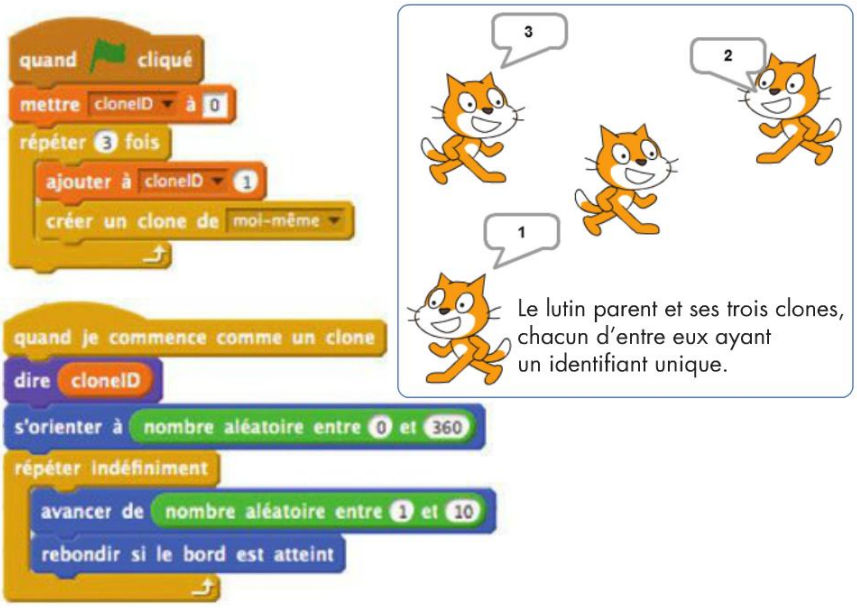


Figure 5-16 : Utilisation de variables pour distinguer les lutins les uns des autres

Le lutin parent de cet exemple possède une variable nommée cloneID. Lorsqu'on clique sur le drapeau vert, il démarre une boucle pour créer trois clones et règle cloneID à une valeur différente (1, 2 et 3 dans ce cas-là) avant de créer un clone. Chaque clone prend vie avec sa propre copie de cloneID initialisée à une valeur différente. Vous pourriez maintenant utiliser un bloc **si**, que nous étudierons en détail dans le chapitre suivant, pour vérifier l'identifiant du clone et lui faire faire l'action correspondante.

ClonesEtVar Globales.sb2

Voyons à présent comment les clones interagissent avec les variables globales. Petit rappel de la figure 5-15 : les variables dont la portée est réglée à Pour tous les lutins peuvent être lues et écrites par la scène et par tous les lutins, y compris les clones. Par exemple, le programme de la figure 5-17 utilise ce fait pour détecter le moment où tous les clones d'un lutin parent ont disparu.



Figure 5-17 : L'utilisation d'une variable globale pour détecter le moment où tous les clones sont supprimés

Dans ce script, le lutin parent initialise la variable globale nbClones à 5 et crée cinq clones. Il attend ensuite que nbClones atteigne 0 pour annoncer la fin du jeu. Les clones apparaissent à des instants et à des lieux aléatoires sur la scène, disent « Bonjour ! » pendant deux secondes, puis disparaissent. Avant qu'un clone soit supprimé, il décrémente nbClones de 1. Lorsque les cinq clones ont disparu, nbClones atteint 0 et le script principal cesse d'attendre ; le lutin original dit alors « Game Over ! »

Copyright © 2016 Eyrolles.

Dans la section suivante, vous étudierez les moniteurs de variables, qui vous montrent les valeurs actuellement stockées dans les variables et vous donnent la possibilité de les modifier. Cette possibilité vous ouvrira la porte vers la création de nouvelles sortes d'applications.

Les moniteurs de variables

Vous serez souvent confronté à l'envie de voir la valeur actuellement stockée dans une variable. Par exemple, lorsqu'un script ne fonctionne pas comme vous le voudriez, vous regarderez l'évolution de certaines de ses variables pour vérifier si elles se comportent correctement. Les moniteurs de variables vous seront d'une grande aide lors d'une telle tâche de débogage.

Les *moniteurs de variables* peuvent être affichés ou cachés de la scène en cochant/décochant la case qui accompagne une variable (figure 5-18). Vous pouvez également contrôler la visibilité d'un moniteur depuis l'intérieur d'un script avec les commandes **montrer la variable** et **cacher la variable**.

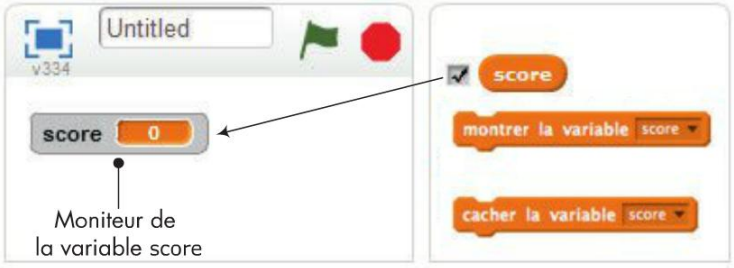


Figure 5-18 : Montrer le moniteur d'une variable en cochant la case qui l'accompagne.

Les moniteurs sont utilisables en lecture ou en potentiomètre. La lecture affiche simplement la valeur contenue dans la variable ; le potentiomètre permet en plus de contrôler directement cette valeur. Double-cliquez sur le moniteur sur la scène pour sélectionner le mode lecture normale (état par défaut), grande lecture, ou potentiomètre. Lorsque vous choisissez ce dernier, vous réglez la plage de contrôle du potentiomètre et sélectionnez ses valeurs minimale et maximale à partir du menu contextuel qui apparaît en cliquant-droit sur l'afficheur (figure 5-19).

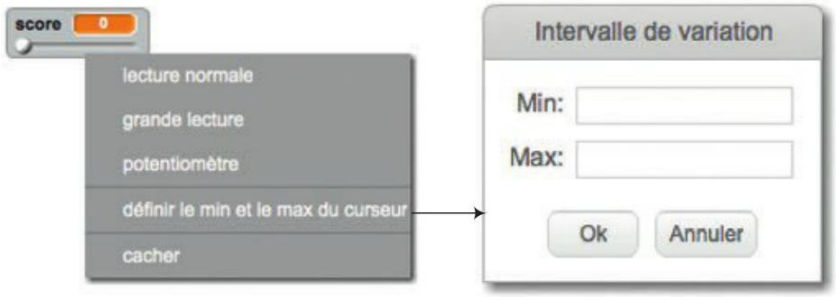


Figure 5-19 : Réglage des valeurs maximale et minimale d'un moniteur en mode potentiomètre

Copyright © 2016 Eyrolles.

CouleurDeScène .sb2

L'utilisation d'un potentiomètre vous permet de changer la valeur d'une variable pendant qu'un script s'exécute, ce qui est un moyen pratique pour les utilisateurs d'interagir avec votre application. Vous pouvez voir un exemple simple de l'utilisation des potentiomètres sur la figure 5-20.

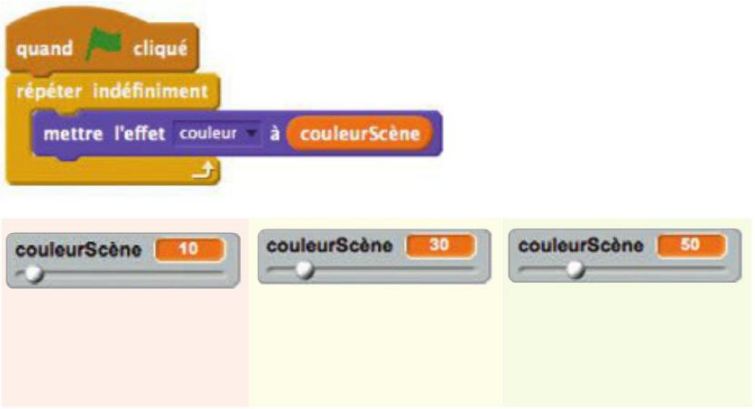
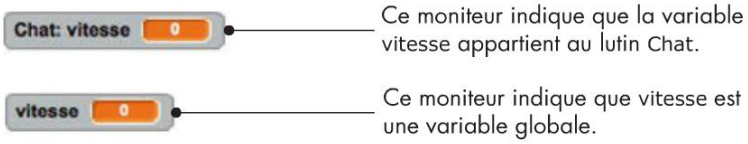


Figure 5-20 : Ajustement de la couleur de la scène avec un potentiomètre

Dans cet exemple, changer la valeur du potentiomètre contrôle la valeur de la variable couleurScène, qui est un paramètre de la commande **mettre l'effet couleur à**. En supposant que ce script appartient à la scène, faire varier le potentiomètre devrait faire varier la couleur de son arrière-plan.

NOTE

Le moniteur d'une variable indique également sa portée. Si une variable appartient à un lutin, son moniteur devrait montrer le nom du lutin avant celui de la variable. Par exemple, le moniteur **Chat vitesse 0** *indique que* vitesse *appartient à* Chat. *Si* vitesse *était une variable globale, alors son moniteur n'indiquerait que* **vitesse 0**. *La différence entre ces deux cas est illustrée sur la figure suivante.*



Utiliser les moniteurs de variables dans des applications

Maintenant que vous connaissez les principes de base des moniteurs de variables, voici quelques exemples d'utilisation qui ajoutent des fonctionnalités à vos applications Scratch.

Pouvoir également utiliser les moniteurs comme contrôleurs ouvre la porte vers la création de nombreux types d'applications, telles que des jeux, des simulations et des programmes interactifs.

Copyright © 2016 Eyrolles.

Simulation de la loi d'Ohm

LoidOhm.sb2

Notre premier exemple est une simulation de la loi d'Ohm. Quand une tension (V) est appliquée à une résistance (R), un courant (I) traverse cette résistance. Selon la loi d'Ohm, la valeur de ce courant est donnée par l'équation suivante :

$$Courant\ (\mathrm{I}) = \frac{Tension\ (\mathrm{V})}{Résistance\ (\mathrm{R})}$$

Notre application permet à l'utilisateur de changer les valeurs de V et de R grâce à des potentiomètres. Elle calcule ensuite la valeur correspondante du courant I et l'affiche. L'interface utilisateur de cette application est présentée à la figure 5-21.

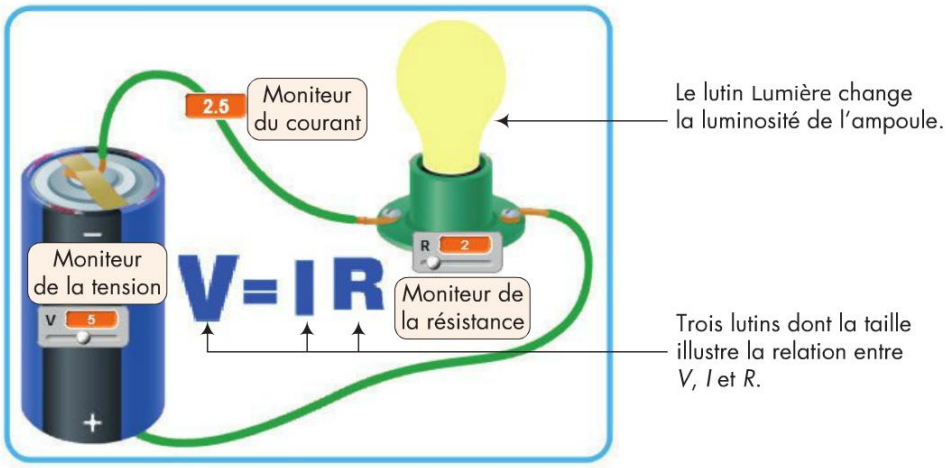


Figure 5-21 : L'interface utilisateur de notre simulateur de loi d'Ohm

Le potentiomètre de la tension de la pile (V) a une plage de contrôle de [0,10] et celui de la résistance (R) a une plage de [1,10]. Lorsque l'utilisateur modifie V ou R avec les potentiomètres, l'application calcule la valeur correspondante du courant (I) qui traverse le circuit. La luminosité de l'ampoule change en fonction de la valeur du courant qui la traverse. Les tailles des lettres V, I et R de la figure changent également selon leurs valeurs respectives.

Cette application possède un total de cinq lutins (nommés Tension, Courant, Résistance, Égal et Lumière) et trois variables (nommées V, I et R). Tout ce que vous voyez d'autre sur la figure 5-21 (la pile, les câbles, les prises, etc.) font partie de l'image d'arrière-plan de la scène. Le script principal qui dirige cette application appartient à la scène et est présenté à la figure 5-22.

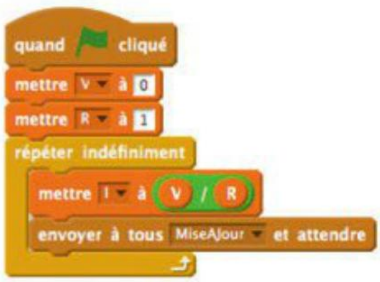


Figure 5-22 : Script principal de l'application Loi d'Ohm

Ce script initialise les valeurs de V et de R puis entre dans une boucle infinie. À chaque passage dans la boucle, il calcule I en utilisant les valeurs actuelles de V et de R, qui sont

Copyright © 2016 Eyrolles.

réglées par l'utilisateur grâce aux potentiomètres. Il envoie ensuite un message aux autres lutins de l'application pour leur indiquer de mettre à jour leur apparence en fonction des valeurs calculées. La figure 5-23 montre les réponses des lutins Tension, Courant, Résistance et Lumière (qui correspondent aux lettres *V*, *I*, *R* et à l'ampoule, respectivement) lorsqu'ils reçoivent le message MiseAJour.

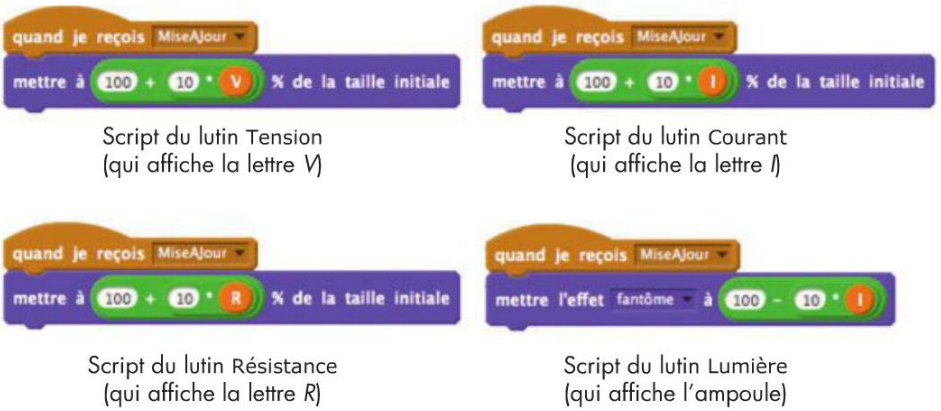


Figure 5-23 : Scripts déclenchés en réponse au message MiseAJour

Lorsque le message MiseAJour est reçu, les lutins Tension, Courant et Résistance changent leur taille (de 100 % à 200 % de leur taille originale) en fonction de la valeur actuelle de leurs variables respectives. Le lutin Lumière exécute la commande **mettre l'effet fantôme à** pour changer son niveau de transparence en fonction de la valeur de I. Cela offre à l'ampoule un effet visuel réaliste.

FAITES LE TEST 5-3

Ouvrez le simulateur de loi d'Ohm pour l'exécuter et étudiez le script pour comprendre comment il fonctionne. À votre avis, que se passerait-il si nous ajoutions la commande **ajouter à l'effet couleur 25** à la fin du script du lutin Lumière ? Implémentez ce changement pour vérifier votre réponse. De quelles manières pourriez-vous améliorer cette application ?

Démonstration d'un circuit en série

CircuitSérie.sb2

Notre deuxième exemple simule un circuit qui contient une pile et trois résistances connectées en série. L'utilisateur peut modifier la tension de la pile ainsi que les valeurs des résistances, grâce à des potentiomètres. Le courant qui traverse les résistances et la tension aux bornes des trois résistances sont affichés en mode grande lecture. Vous pouvez voir l'interface de cette application à la figure 5-24 (notez que la couleur des bandes des résistances ne représente pas leur valeur réelle dans notre simulation).

Copyright © 2016 Eyrolles.

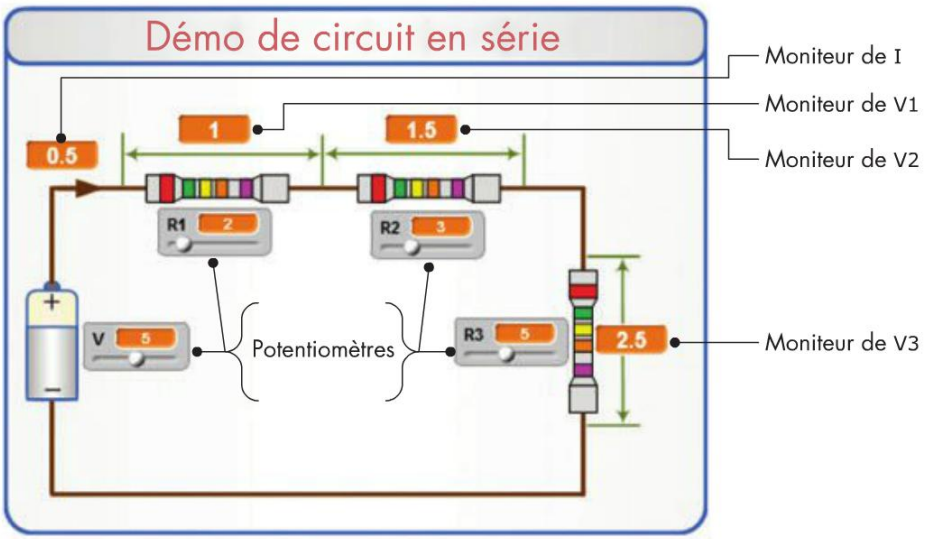


Figure 5-24 : Une application qui fait la démonstration d'un circuit en série

Les équations qui gouvernent le fonctionnement de ce circuit sont présentées ci-après. Nous pouvons calculer le courant qui traverse le circuit en divisant la tension V de la pile par la somme des trois résistances. Ensuite, la tension aux bornes de chaque résistance est calculée en multipliant le courant par la valeur de la résistance :

$$\text{Résistance totale : } R_{tot} = R_1 + R_2 + R_3$$
$$\text{Courant qui traverse le circuit : } I = V / R_{tot}$$
$$\text{Tension aux bornes de } R_1 : V_1 = I \times R_1$$
$$\text{Tension aux bornes de } R_2 : V_2 = I \times R_2$$
$$\text{Tension aux bornes de } R_3 : V_3 = I \times R_3$$

Cette application ne possède pas de lutin mais, lorsqu'on clique sur le drapeau vert, le script de la figure 5-25, qui appartient à la scène, est exécuté.

Ce script effectue les calculs et affiche les résultats sur la scène. Notez que, même si les potentiomètres de R2 et R3 peuvent varier de 0 à 10, la valeur minimale de R1 a été intentionnellement réglée à 1. Cela nous assure que Rtot soit toujours strictement positive et évite une division par 0 lors du calcul du courant.

La plus grande partie du travail sur cette application a été consacrée à la conception de l'interface (c'est-à-dire l'arrière-plan de la scène). Ceci fait, nous n'avions plus qu'à positionner les moniteurs au bon endroit sur la scène.

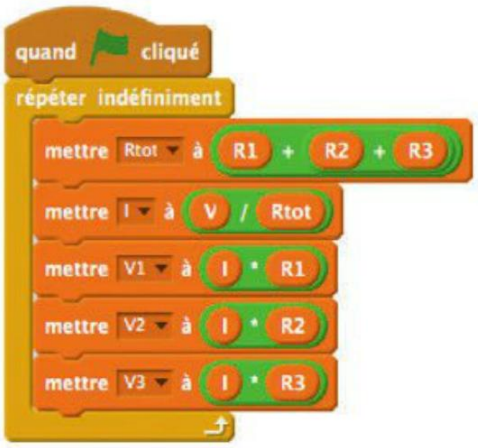


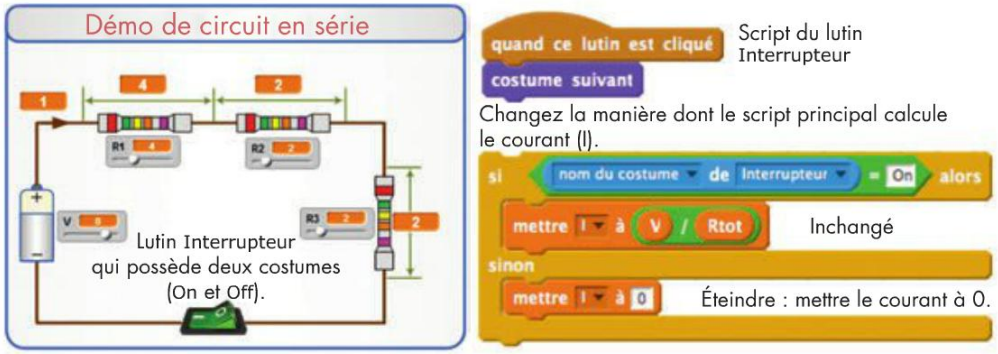
Figure 5-25 : Le script exécuté lors d'un clic sur le drapeau vert

Copyright © 2016 Eyrolles.

CircuitSérie Interrupteur.sb2

FAITES LE TEST 5-4

Ouvrez le simulateur de circuit en série et exécutez-le. Testez différentes valeurs pour R1, R2, R3 et V. Observez les valeurs calculées de V1, V2 et V3 pendant que vous agissez sur les potentiomètres. Quelle relation lie la somme des tensions (V1 + V2 + V3) et la tension de la pile ? Qu'est-ce que cela vous apprend sur la relation entre les tensions dans un circuit en série ? Vous apporterez une amélioration intéressante à cette application en y ajoutant l'image d'un interrupteur qui ouvre ou ferme le circuit, comme présenté ci-après. Quand l'interrupteur est ouvert, aucun courant ne traverse le circuit. Essayez d'implémenter ce changement en utilisant les indices donnés sur la figure.



Visualiser le volume et l'aire d'une sphère

Sphère.sb2

Notre troisième exemple est une application interactive qui calcule le volume et l'aire d'une sphère. On peut y modifier le diamètre de la sphère en cliquant sur les boutons de l'interface utilisateur. L'application calcule et affiche automatiquement le volume et l'aire correspondants.

Pour rendre cette application plus attrayante, la taille de la sphère affichée sur la scène change selon le diamètre sélectionné. L'interface utilisateur de cette application est illustrée à la figure 5-26.

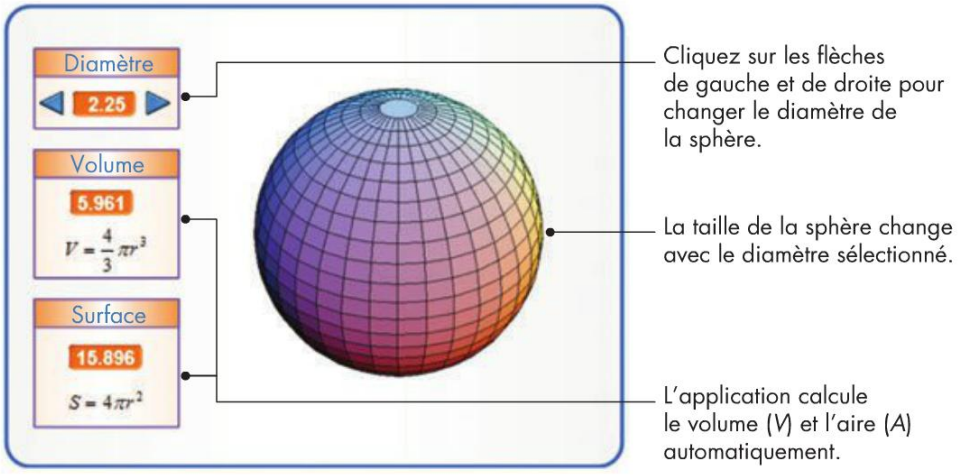


Figure 5-26 : Interface utilisateur de l'application sphère

Copyright © 2016 Eyrolles.

Cette application contient trois lutins : les deux flèches (nommées Plus et Moins) et l'image de la sphère (nommée Sphère). Les scripts associés aux deux boutons envoient un message pour indiquer qu'on a cliqué dessus (figure 5-27).

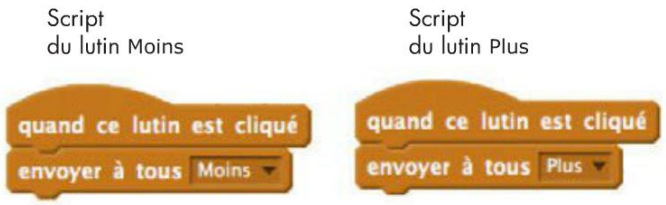


Figure 5-27 : Scripts des lutins Plus *et* Moins

Le lutin Sphère possède neuf costumes qui représentent les diamètres de la sphère 1, 1.25, 1.5, 1.75 … 3. Quand ce lutin reçoit le message Plus ou Moins, il exécute les scripts de la figure 5-28.

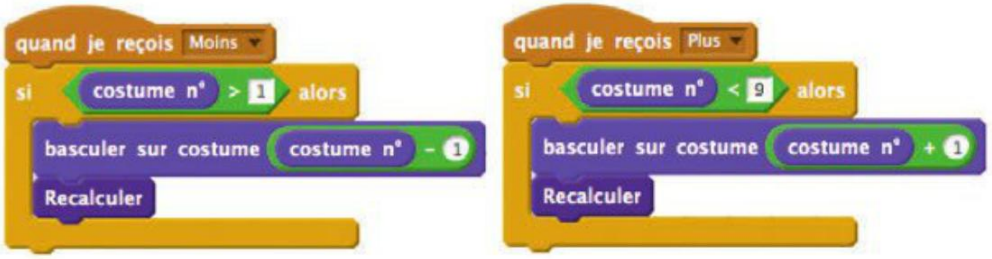


Figure 5-28 : Scripts déclenchés par les messages Plus *et* Moins

Le lutin change de costume puis appelle la procédure **Recalculer** pour mettre à jour l'aire et le volume. Notez que ces scripts utilisent la valeur du costume actuel pour déterminer si la sphère a atteint sa taille maximale ou minimale, assurant ainsi la validité des réponses aux boutons Plus et Moins. Je parlerai du bloc **si** plus en détail dans le chapitre suivant mais, pour l'instant, concentrons-nous sur la procédure **Recalculer** présentée à la figure 5-29.

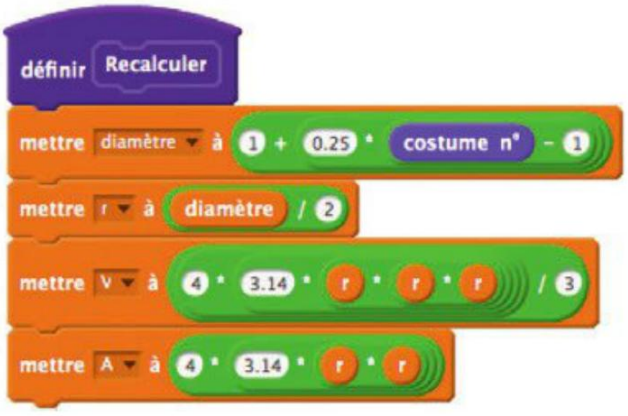


Figure 5-29 : La procédure **Recalculer**

Tout d'abord, la valeur de la variable diamètre est réglée selon la formule suivante :

$$diamètre = 1 + 0{,}25 \times (numéro\ de\ costume - 1)$$

Copyright © 2016 Eyrolles.

Puisque le numéro de costume varie de 1 à 9, les valeurs correspondantes de la variable diamètre seront 1, 1,25, 1,50 … 2,75, 3, ce qui correspond à ce que nous voulions.

Le script calcule la valeur du rayon *r* en divisant le diamètre par 2. Il calcule ensuite le volume et l'aire de la sphère en utilisant les formules de la figure 5-26. Les valeurs calculées sont affichées automatiquement sur le moniteur correspondant sur la scène.

FAITES LE TEST 5-5

Ouvrez l'application et exécutez-la. Étudiez ses scripts pour comprendre comment elle fonctionne. Ajoutez un script au lutin Sphère afin qu'il tourne et change de couleur pendant l'exécution. Comme second exercice, modifiez le programme original pour utiliser un costume unique pour le lutin Sphère et utilisez le bloc **ajouter à la taille** pour changer la taille de la sphère. L'image redimensionnée ne sera pas aussi jolie, mais hormis ce détail, l'application devrait s'exécuter de façon identique.

Dessiner une rosace à n pétales

RoseNPétales.sb2

Dans cet exemple, vous allez créer une application qui dessine une rosace avec plusieurs pétales sur la scène. La procédure de dessin de rosace peut être divisée en plusieurs étapes :

1. Commencer au point d'origine de la scène.
2. Orienter le lutin dans une direction. Par convention, la lettre grecque θ (prononcée *thêta*) représente un angle ; nous appellerons donc thêta la variable correspondant à la direction du lutin.
3. Déplacer le lutin de r pas et dessiner un point sur la scène. Ensuite, lever le stylo et retourner au point d'origine.
4. Changer l'angle thêta d'une certaine valeur (nous prendrons 1) et répéter les étapes 2 à 4.
   La relation qui lie la distance r et l'angle thêta est la suivante :

$$r = a \times \cos(n \times \theta)$$

où a est un nombre réel et n est un entier. Cette équation produit une rosace dont la taille et le nombre de pétales sont contrôlés par a et n, respectivement. Cette équation utilise également la fonction trigonométrique cosinus (*cos*), que vous pourrez trouver dans un bloc rapporteur de la palette *Opérateurs* (regardez le bloc **racine de**). Étant données les valeurs de a et n, il ne nous reste plus qu'à choisir des valeurs différentes de thêta, calculer les valeurs correspondantes de r et marquer les points résultants sur la scène (figure 5-30).

Copyright © 2016 Eyrolles.

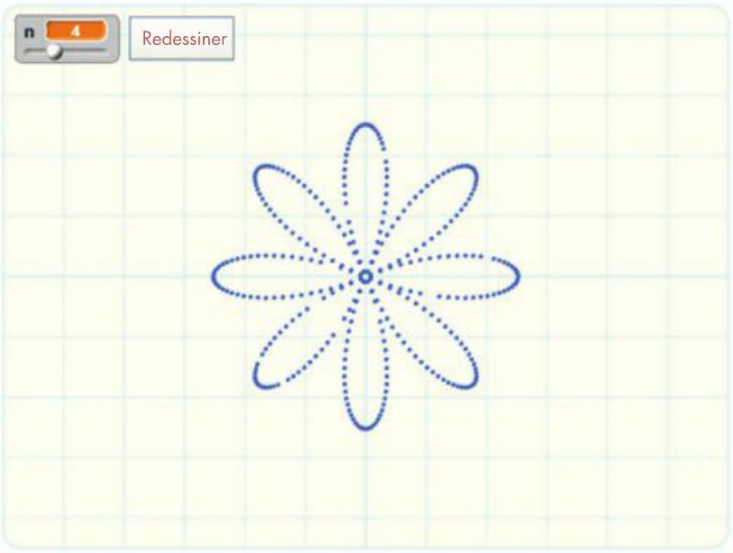


Figure 5-30 : L'interface utilisateur de l'application rosace à n *pétales*

Cette application contient deux lutins : le premier possède le costume du bouton Redessiner, le second (appelé Peintre), est caché et dessine la rosace. L'utilisateur contrôle le nombre de feuilles dessinées en changeant n grâce au potentiomètre et en cliquant sur le bouton Redessiner pour tracer la rosace correspondante. Quand l'utilisateur clique sur ce bouton, le lutin bouton envoie un message Redessiner, auquel le lutin Peintre répond en exécutant le script de la figure 5-31.

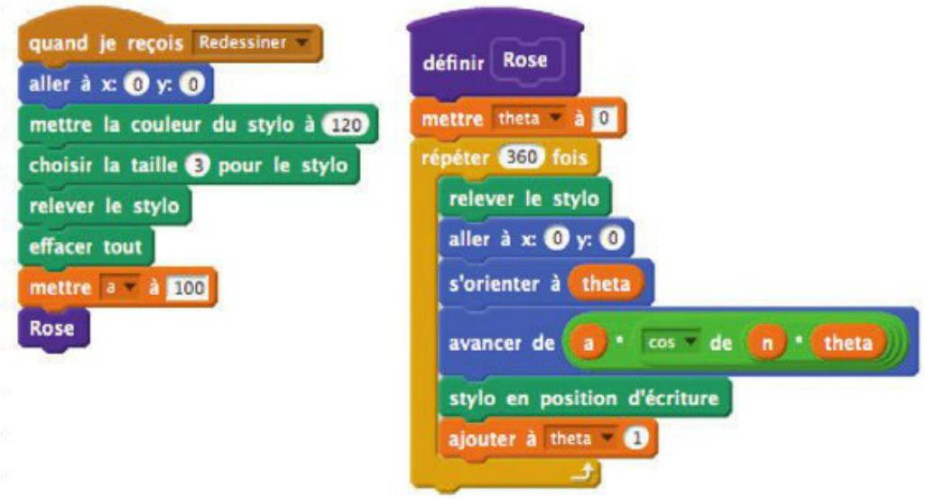


Figure 5-31 : La procédure **Redessiner** *qui trace la rosace à* n *pétales sur la scène*

Tout d'abord, le lutin initialise sa couleur et sa taille, puis efface les traces de stylo précédentes de la scène. Ensuite, il règle la variable a à 100 et appelle la procédure **Rosace**, qui exécute une boucle 360 fois pour dessiner la rosace sur la scène. À chaque passage dans la boucle, la procédure oriente le lutin vers thêta, le déplace de r pas et dessine un point à cet endroit. Elle incrémente ensuite de 1 la valeur de thêta pour se préparer au prochain passage dans la boucle **répéter**.

Copyright © 2016 Eyrolles.

La figure 5-32 montre quelques rosaces créées avec différentes valeurs de n.

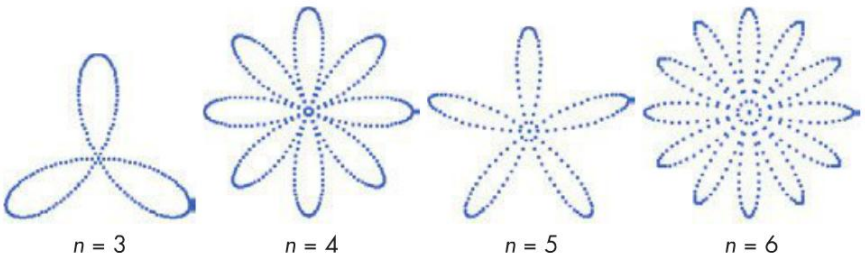


Figure 5-32 : Quelques rosaces créées avec la procédure **Rosace**

FAITES LE TEST 5-6

Ouvrez l'application et exécutez-la. Changez la valeur de n pour voir ce que vous pouvez créer d'autre avec la procédure **Rosace**. Ajoutez un autre potentiomètre à l'application pour permettre à l'utilisateur de changer la valeur de a et modifiez le script comme il se doit. Vous pouvez également modifier la procédure **Rosace** pour qu'elle prenne a comme paramètre.

Modéliser la répartition des graines de tournesol

Tournesol.sb2

Des biologistes et des mathématiciens ont intensément étudié la disposition des feuilles sur les tiges de plantes. À notre tour de nous plonger un peu dans la botanique en étudiant les modèles géométriques qui représentent les fleurs dont la répartition des graines est en spirale. Nous programmerons plus particulièrement deux équations qui modélisent la répartition des graines dans un tournesol. Pour dessiner la *n*ième graine d'un tournesol, nous suivrons les étapes suivantes :

1. Orienter le lutin dans la direction $n \times 137{,}5°$.
2. Faire avancer le lutin d'une distance $r = c\sqrt{n}$, où c est un facteur d'échelle constant (réglé à 5 dans notre exemple).
3. Dessiner un point sur la scène une fois cette position atteinte.

Nous répéterons ces étapes pour chaque graine que nous voudrons dessiner. Pour la première graine, nous réglerons $n = 1$; pour la seconde, nous réglerons $n = 2$ et ainsi de suite. Utiliser des angles différents de 137,5° dans la première étape donnera différents types de répartition des graines.

NOTE

Si vous êtes curieux et si vous voulez en savoir plus sur les équations modélisant la répartition des graines dans un tournesol, je vous invite à consulter le livre (en anglais) The Algorithmic Beauty of Plants (La beauté algorithmique des plantes) *de Przemyslaw Prusinkievicz et Aristid Lindenmayer (Springer-Verlag, 2004), particulièrement le chapitre 4, que vous trouverez sur le site web de l'ouvrage,* http://algorithmicbotany.org/papers/#abop.

Copyright © 2016 Eyrolles.

Notre application générera des motifs similaires à ceux décrits dans ces travaux, dont quelques exemples sont visibles sur la figure 5-33.

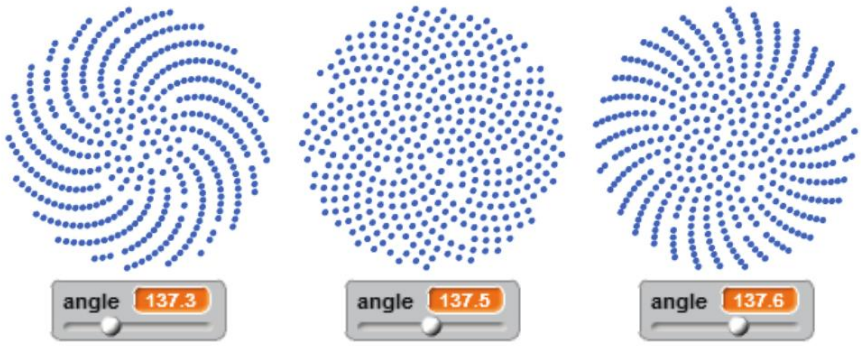


Figure 5-33 : Des tournesols générés avec différents angles

L'interface de cet exemple contient un potentiomètre qui permet de changer la valeur de l'angle de 137° à 138° avec un pas de 0,01° et un bouton nommé Redessiner. Quand l'utilisateur clique sur ce bouton, le lutin correspondant envoie un message au lutin Peintre, qui exécute alors le script présenté à la figure 5-34. La procédure **Tournesol** exécute une boucle qui dessine 420 graines. Vous pouvez changer ce nombre si vous le désirez. À chaque itération de la boucle, la procédure déplace le lutin au lieu de la $n$ième graine (en calculant l'angle de la graine ❶ et en déplaçant le lutin de $c\sqrt{n}$ pas ❷) et dessine un point à cet endroit. Ensuite, la procédure incrémente n, qui représente le numéro de graine, pour se préparer à dessiner la prochaine.

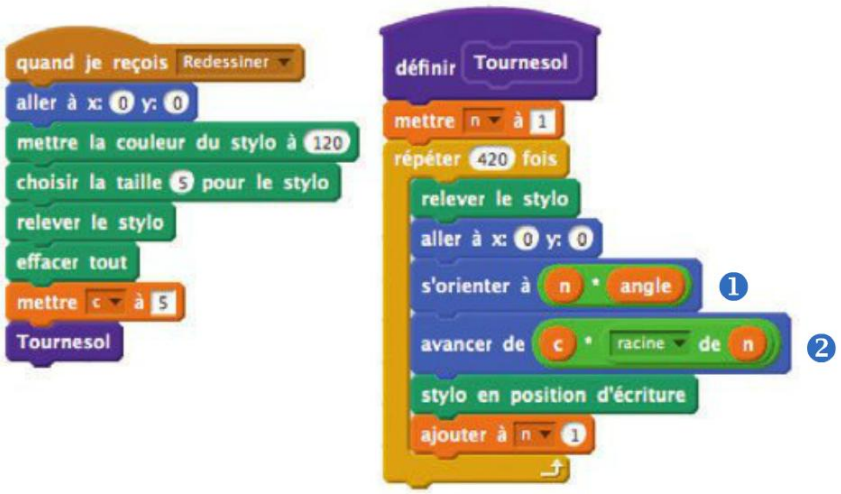


Figure 5-34 : Le script du lutin Peintre

Les scripts que je vous ai présentés dans cette section ne sont que quelques exemples parmi le grand nombre d'applications géniales que nous pouvons créer en utilisant les moniteurs de variables. Permettre à l'utilisateur d'interagir avec nos applications grâce à des potentiomètres ne représente qu'une partie des moyens de rendre des applications interactives. Dans la section suivante, vous apprendrez à créer des scripts qui demandent à l'utilisateur d'entrer des données dans vos programmes.

Copyright © 2016 Eyrolles.

FAITES LE TEST 5-7

Ouvrez l'application et exécutez-la. Changez la valeur de l'angle pour voir ce que vous pouvez créer avec la procédure **Tournesol**. Étudiez la procédure pour comprendre comment elle fonctionne et essayez de trouver des idées pour l'améliorer.

Obtenir des données entrées par les utilisateurs

EntréesUtilisateur .sb2

Imaginons que vous vouliez créer un jeu qui apprenne aux enfants les bases de l'arithmétique. Votre jeu contiendrait probablement un lutin qui afficherait une addition et demanderait au joueur d'entrer le résultat. Comment lirez-vous la réponse qu'entrera le joueur pour vérifier si elle est correcte ?

La palette *Capteurs* de Scratch fournit un bloc de commande, **demander et attendre**, à utiliser pour obtenir des données entrées par l'utilisateur. Ce bloc prend un seul paramètre d'entrée qui correspond à la chaîne de caractères montrée à l'utilisateur, généralement sous forme de question. Comme le montre la figure 5-35, l'exécution de ce bloc produit des résultats différents selon la visibilité du lutin (si le lutin est montré ou caché). La sortie montrée à droite apparaît également lorsque la commande **demander et attendre** est appelée par la scène.

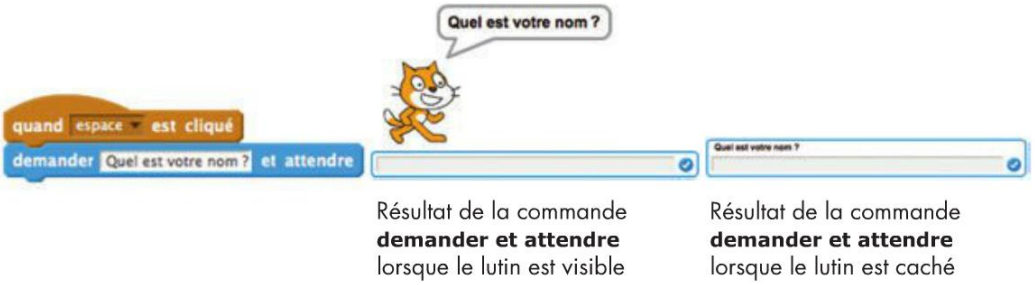


Figure 5-35 : Le bloc **demander et attendre** *produit différents résultats selon la visibilité du lutin.*

Après avoir exécuté la commande **demander et attendre**, le script appelant attend que l'utilisateur appuie sur la touche Entrée ou clique sur le bouton bleu à droite de la zone de texte. Scratch stocke alors la réponse de l'utilisateur dans le bloc **réponse** et reprend l'exécution par la commande qui suit le bloc **demander et attendre**. Pour voir cette commande en action, jetez un œil aux exemples suivants.

Lire un nombre

DemanderEt Attendre.sb2

Le script de la figure 5-36 demande son âge à l'utilisateur, attend qu'il réponde et lui indique quel âge il aura dans 10 ans.

Copyright © 2016 Eyrolles.



Figure 5-36 : Un script qui prend l'âge de l'utilisateur en entrée

La figure montre la sortie du programme lorsque l'utilisateur tape `18` et appuie sur la touche Entrée du clavier. Vous remarquerez que le programme utilise le bloc **regroupe** (de la palette *Opérateurs*) pour concaténer (ou connecter) deux chaînes de caractères.

Lire des caractères

DemanderEt Attendre2.sb2

Le script de la figure 5-37 demande ses initiales à l'utilisateur pour créer et afficher un message de salutation à son intention et basé sur sa réponse.



Figure 5-37 : Un script qui utilise deux variables pour lire et stocker les initiales de l'utilisateur

Ce programme utilise deux variables (`premièreInitiale` et `secondeInitiale`) pour enregistrer les valeurs entrées par l'utilisateur. Vous voyez la sortie finale du programme lorsque l'utilisateur entre les initiales `M` et `S`. Remarquez que le programme utiliser des blocs **regroupe** imbriqués pour construire le message de salutation. Cette technique est utilisable pour créer toutes sortes de chaînes de caractères et afficher des messages personnalisés dans vos applications.

Effectuer des opérations arithmétiques

DemanderEt Attendre3.sb2

Le script de la figure 5-38 demande à l'utilisateur de rentrer deux nombres. Il en calcule le produit et affiche le résultat dans une bulle grâce à la commande **dire**. Comme dans l'exemple précédent, le script utilise deux variables (`num1` et `num2`) pour stocker les valeurs entrées par l'utilisateur.

Copyright © 2016 Eyrolles.



Figure 5-38 : Calcul d'une valeur en fonction des entrées de l'utilisateur

La figure montre la sortie du programme lorsque l'utilisateur entre les nombres 9 et 8. Vous remarquerez à nouveau les blocs **regroupe** imbriqués pour construire la chaîne de caractères de sortie.

Les exemples présentés dans cette section montrent plusieurs manières d'utiliser le bloc **demander et attendre** pour écrire des scripts qui prennent en compte les entrées de l'utilisateur et résolvent des problèmes variés. Vous pouvez par exemple écrire un programme qui trouve les racines d'une équation du second degré de la forme $ax^2 + bx + c = 0$ pour n'importe quelles valeurs de a, b et c entrées par l'utilisateur. Vous pourriez ensuite utiliser ce programme pour vérifier vos propres réponses à ce type d'équations. J'espère que cela vous donnera des idées sur la façon d'utiliser ce bloc puissant pour résoudre n'importe quel problème mathématique que vous pourriez rencontrer.

Résumé

Les variables font partie des concepts les plus importants de la programmation. Une variable est le nom d'une zone de la mémoire de l'ordinateur dans laquelle nous pouvons stocker une donnée, telle qu'un nombre ou une chaîne de caractères.

Dans ce chapitre, vous avez appris quels étaient les types de données basiques acceptés par Scratch et les opérations autorisées avec ces types. Vous avez ensuite appris comment créer des variables et comment les utiliser pour stocker des données.

Vous avez également implémenté plusieurs applications pratiques qui utilisaient des variables pour montrer différentes fonctions. Vous avez étudié les moniteurs de variables et les avez utilisés pour créer différentes sortes de programmes interactifs. Enfin, vous avez appris à utiliser le bloc **demander et attendre** pour demander à l'utilisateur d'entrer des réponses afin de les traiter ensuite.

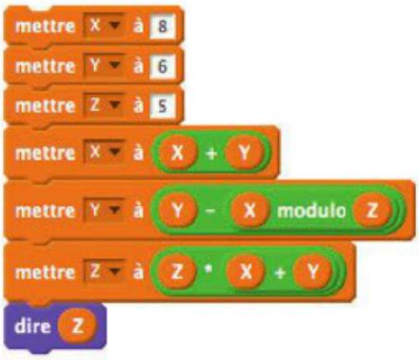
Dans le chapitre suivant, vous en apprendrez plus sur ce que sont les données booléennes et sur le rôle fondamental qu'elles jouent dans les prises de décision. Vous étudierez également les blocs **si** et **si/alors** et les utiliserez pour améliorer l'intelligence de vos programmes Scratch.

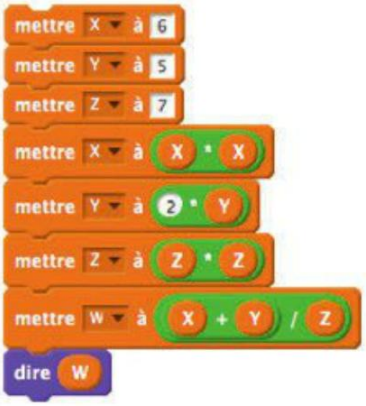
Copyright © 2016 Eyrolles.

Problèmes

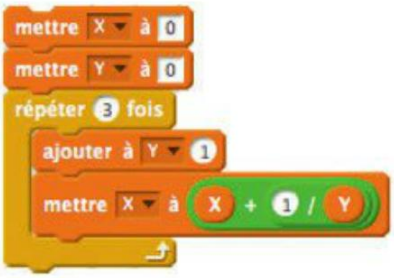
1. Créez un script qui implémente les instructions suivantes :
   - Régler la variable vitesse à 60 km/h.
   - Régler la variable temps à 2,5 heures.
   - Calculer la distance parcourue et enregistrer la réponse dans la variable distance.
   - Afficher la distance calculée, avec un message approprié.
2. Quelle sera la sortie de chacun des scripts suivants ? Reproduisez-les et exécutez-les pour vérifier vos réponses.







3. Quelles seront les valeurs de X et Y à la fin de chaque itération de la boucle du script suivant ? Reproduisez ce dernier et exécutez-le pour vérifier votre réponse.



4. Soient *x* et *y* deux variables. Créez des blocs de fonction équivalents aux déclarations suivantes :
   - Ajouter 5 à *x* et stocker le résultat dans *y*.
   - Multiplier *x* par 3 et stocker le résultat dans *y*.
   - Diviser *x* par 10 et stocker le résultat dans *y*.
   - Soustraire 4 à x et stocker le résultat dans *y*.
   - Mettre *x* au carré, ajouter *y* et stocker le résultat dans *x*.
   - Régler *x* à deux fois la valeur de *y* plus trois fois le cube de *y*.
   - Régler *x* à moins *y* au carré.
   - Régler *x* au résultat de la division de la somme de *x* et *y* par le produit de *x* et *y*.
5. Écrivez un programme qui demande à l'utilisateur de rentrer un article, un nom et un verbe. Le programme créera alors une phrase de la forme *article nom verbe*.

Copyright © 2016 Eyrolles.

6. Écrivez un programme qui demande à l'utilisateur d'entrer une température en degrés Fahrenheit. Le programme convertira la température en degrés Celsius et affichera le résultat avec un message approprié. Indice : $C° = (F° \times 32) \ / \ 1{,}8$

7. Quand le courant I traverse une résistance R, la puissance P dissipée dans la résistance est de $I^2 \times R$. Écrivez un programme qui lise I et R et calcule P.

8. Écrivez un programme qui lise la longueur des deux côtés d'un triangle rectangle et calcule la longueur de l'hypoténuse.

9. Écrivez un programme qui demande à l'utilisateur d'entrer la longueur (L), la largeur (l) et la hauteur (H) d'une boîte. Le programme calculera et affichera ensuite le volume et l'aire de la boîte.

$$Volume = L \times l \times H$$

$$aire = 2 \times [(L \times l) + (L \times H) + (H \times l)]$$

10. La résistance équivalente R de trois résistances R_1, R_2 et R_3 montées en parallèle est donnée par l'équation suivante :

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

Écrivez un programme qui lise les valeurs de R_1, R_2 et R_3 puis calcule R.

Tape-Taupe.sb2

11. Complétez le jeu de Tape-taupe présenté dans ce chapitre. Le fichier *Tape-Taupe.sb2* contient une implémentation partielle de ce programme. Lorsqu'on clique sur le drapeau vert, le script fourni démarre une boucle qui déplace le lutin Chat aléatoirement sur les trous. Ajoutez deux scripts (un pour le Chat et un autre pour la scène) qui changeront les valeurs des deux variables (touché et raté) de la bonne manière. Essayez d'y ajouter des effets sonores pour rendre le jeu plus amusant ! Vous pouvez également ajouter une condition qui termine le jeu au bout d'un certain temps ou lorsque la variable raté atteint une certaine valeur.



Copyright © 2016 Eyrolles.

6

PRENDRE DES DÉCISIONS

Ce chapitre va vous apprendre quels outils de Scratch utiliser pour écrire des programmes pouvant comparer des valeurs, évaluer des expressions logiques et prendre des décisions en fonction de ces résultats. Il vous présentera également plusieurs exemples d'application. Voici ce que vous allez découvrir :

- des techniques de résolution de problèmes de base ;
- comment utiliser les blocs **si** et **si/sinon** pour choisir parmi plusieurs actions possibles ;
- comment construire des expressions logiques pour évaluer des conditions données ;
- comment contrôler le chemin emprunté par un programme dans des expressions à branches.

Les programmes que vous avez écrits jusqu'à présent suivent un modèle d'exécution simple. Ils commencent par la première instruction, l'exécutent,

Copyright © 2016 Eyrolles.

passent à l'instruction suivante, et ce, jusqu'à atteindre la fin du programme. Les blocs de commande de ces programmes sont exécutés en séquence, sans ignorer ou sauter une partie du programme.

Cependant, le besoin d'utiliser un chemin d'exécution différent du chemin séquentiel habituel est très fréquent en programmation. Si vous écriviez une application censée apprendre les bases de l'arithmétique à des enfants, vous voudriez exécuter certains blocs pour récompenser les bonnes réponses et un ensemble de blocs complètement différent pour les mauvaises réponses (pour dévoiler la bonne réponse ou proposer une nouvelle chance, par exemple). Votre script peut décider de ce qu'il doit faire en comparant le résultat entré par l'élève et la réponse correcte. Ceci constitue la base de toute prise de décision.

Dans ce chapitre, vous étudierez les commandes de prise de décision disponibles dans Scratch et vous écrirez plusieurs programmes utilisant ces commandes pour vérifier les données entrées par l'utilisateur et accomplir différentes actions.

Tout d'abord, je vous présenterai les opérateurs de comparaison de Scratch et vous montrerai comment les utiliser pour comparer des nombres, des lettres et des chaînes de caractères. Ensuite, je vous présenterai les blocs **si** et **si/sinon** et vous expliquerai le rôle majeur qu'ils jouent dans la prise de décision. Vous apprendrez également comment tester plusieurs conditions en utilisant des blocs **si** et **si/sinon** imbriqués et vous écrirez des programmes pilotés par des menus pour mettre ces blocs en action. Ensuite, je vous présenterai les opérateurs logiques, qui offrent une autre manière de tester plusieurs conditions. Dans la dernière section, nous écrirons plusieurs programmes intéressants basés sur tous les concepts que vous avez appris jusqu'à présent.

Les opérateurs de comparaison

Vous prenez des décisions tous les jours et chaque décision vous mène à accomplir ou non certaines actions. Vous pourriez par exemple vous dire « si cette voiture vaut moins de 2 000 €, je l'achète » ; vous demanderez alors le prix de la voiture et déciderez ou non de l'acheter.

Il est possible de prendre des décisions dans Scratch. Les *opérateurs de comparaison* servent à comparer les valeurs de deux variables ou expressions et à déterminer si l'une est plus grande, plus petite ou égale à l'autre. Ils sont aussi appelés *opérateurs relationnels* car ils testent la relation qui lie deux valeurs. Les trois opérateurs relationnels proposés par Scratch sont présentés dans le tableau 6-1.

Copyright © 2016 Eyrolles.

Tableau 6-1 : Opérateurs relationnels de Scratch

Opérateur	Signification	Exemple
	Plus grand que	 Le prix est-il plus grand que 2000 ?
	Plus petit que	 Le prix est-il plus petit que 2000 ?
	Égal à	 Le prix est-il égal à 2000 ?

LES BOOLÉENS DANS LE MONDE RÉEL

Le mot *booléen* est utilisé en l'honneur de George Boole, un mathématicien anglais du XIX^{e} siècle qui inventa un système de logique basé sur deux valeurs : 1 et 0 (ou Vrai et Faux, *True* et *False* en anglais). L'algèbre booléenne devint ensuite la base de l'informatique moderne.

Dans la vie réelle, nous employons sans arrêt des expressions booléennes pour prendre des décisions. Les ordinateurs les utilisent également pour décider quelle branche d'un programme suivre. Par exemple, un bras robotisé pourrait être programmé pour inspecter les pièces d'une chaîne d'assemblage et déplacer chaque pièce dans la Benne 1 si bonneQualité = vrai, ou dans Benne 2 si bonneQualité = faux. Les systèmes de sécurité d'habitations sont généralement programmés pour déclencher une alarme si le mauvais code est entré (codeCorrect = faux) ou pour la désactiver si le bon code est entré (codeCorrect = vrai). L'ordinateur de bord de votre voiture déploiera l'airbag automatiquement s'il décide qu'une collision a eu lieu (collision = vrai). Votre téléphone portable peut afficher une icône d'avertissement lorsque la batterie est faible (batterieFaible = vrai) et enlever cette icône lorsque la charge est acceptable (batterieFaible = faux).

Ce ne sont que quelques exemples de la façon dont les ordinateurs déclenchent différentes actions en testant les résultats de conditions booléennes.

Vous avez sans doute remarqué que tous les blocs du tableau 6-1 ont des formes hexagonales. Cela signifie que le résultat de l'évaluation d'un tel bloc donne une valeur *booléenne*, qui est soit vraie soit fausse. Ces expressions sont donc également appelées *expressions booléennes*.

Par exemple, l'expression **prix < 2000** vérifie si la valeur de la variable prix est inférieure à 2 000. Si c'est le cas, le bloc renvoie la valeur vrai. Sinon, il renvoie faux. Vous pouvez utiliser cette expression pour mettre en place une condition de décision de la forme « si (**prix < 2000**), alors acheter une voiture ».

NOTE *Dans Scratch, le résultat d'un opérateur booléen n'a pas été traduit de l'anglais et il renverra les réponses true et false au lieu de vrai et faux, respectivement. Pour la clarté, nous garderons la version traduite en français dans ce livre.*

Copyright © 2016 Eyrolles.

Avant de nous pencher sur le bloc **si**, qui sert à implémenter de tels tests, abordons un exemple simple qui illustre la façon dont les expressions booléennes sont évaluées dans Scratch.

Évaluation des expressions booléennes

Imaginons que nous réglions deux variables x et y de la façon suivante : x = 5 et y = 10. Le tableau 6-2 montre des exemples qui utilisent les blocs relationnels de Scratch.

Ces exemples mettent en évidence des points importants sur les opérateurs relationnels. Tout d'abord, nous pouvons aussi bien les utiliser pour comparer deux variables individuelles (telles que x et y) que pour comparer des expressions (telles que 2 * x et x + 6). Ensuite, le résultat d'une comparaison est toujours vrai ou faux, puisque c'est une valeur booléenne. Enfin, le bloc **x = y** ne signifie pas « mettre x égal à y », mais « est-ce que x est égal à y ? » Donc, lorsque l'expression **mettre z à (x = y)** est exécutée, x est toujours égal à 5.

Tableau 6-2 : Quelques exemples d'utilisation des blocs relationnels

Expression	Signification	z (sortie)	Explication
	z = est-ce que (5 < 10) ?	z = vrai	Car 5 est inférieur à 10
	z = est-ce que (5 > 10) ?	z = faux	Car 5 n'est pas supérieur à 10
	z = est-ce que (5 = 10) ?	z = faux	Car 5 n'est pas égal à 10
	z = est-ce que (10 > 2*5) ?	z = faux	Car 10 n'est pas supérieur à 10
	z = est-ce que (5 = 5) ?	z = vrai	Car 5 est égal à 5
	z = est-ce que (10 < 5 + 6) ?	z = vrai	Car 10 est inférieur à 11

Comparer des lettres et des chaînes de caractères

Imaginons un jeu dans lequel le joueur essaie de deviner un code secret d'une lettre comprise entre *A* et *Z*. Le jeu lit la réponse du joueur, la compare au code secret et indique au joueur de raffiner sa réponse par rapport à l'ordre alphabétique. Par exemple, si la lettre secrète est *G* et si le joueur entre *B*, le jeu dit quelque chose comme « après B » pour indiquer au joueur que la lettre secrète se trouve plus loin dans l'alphabet. Comment comparer la lettre correcte avec celle entrée par le joueur pour décider du message à afficher ?

Heureusement, les opérateurs relationnels de Scratch savent également comparer des lettres. Comme le montre la figure 6-1, Scratch compare les lettres en fonction de l'ordre alphabétique. Puisque la lettre *A* est positionnée avant la lettre *B* dans l'alphabet, l'expression **A < B** renvoie vrai. Cependant, il est important de noter que ces comparaisons ne prennent pas en compte les majuscules. La lettre majuscule *A* est considérée identique à

Copyright © 2016 Eyrolles.

la lettre minuscule *a*. L'expression **A = a** renvoie donc vrai. Par ailleurs, les lettres comportant un accent ou une cédille (telles que *é* ou *ç*) sont considérées comme inférieures aux lettres normales. En prenant la lettre *e* comme exemple, l'ordre des accents est le suivant : *è* < *é* < *ê* < *ë*. Les lettres à accent ou cédille sont comparées entre elles dans l'ordre alphabétique également. D'autre part, les chiffres sont considérés inférieurs aux lettres et les caractères spéciaux (tels que & : ; / @…) sont considérés inférieurs aux chiffres.

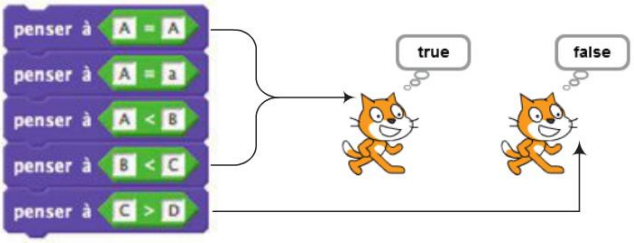


Figure 6-1 : Utilisation des opérateurs relationnels pour comparer des lettres

Connaissant ces informations, vous pouvez tester la réponse du joueur en utilisant l'ensemble des conditions suivantes :

```
SI (réponse = codeSecret), alors dire Correct
SI (réponse > codeSecret), alors dire Avant <réponse>
SI (réponse < codeSecret), alors dire Après <réponse>
```

Une expression *conditionnelle* a la forme suivante : « si la condition est vraie, alors accomplir cette action ». Dans la section suivante, je vous apprendrai à implémenter des conditions dans Scratch, mais étudions d'abord les opérateurs relationnels de façon plus approfondie avec notre jeu de code secret.

Et si le code secret contenait plus d'une lettre ? Le joueur pourrait par exemple avoir à deviner le nom d'un animal. L'utilisation des opérateurs relationnels de Scratch est-elle toujours possible ? Par chance, la réponse est oui : vous pouvez utiliser les opérateurs relationnels de Scratch pour comparer des chaînes de caractères. Cependant, comment Scratch évaluera-t-il une comparaison du type **girafe > souris ?** Les exemples de la figure 6-2 illustrent le résultat d'une comparaison de chaînes de caractères.

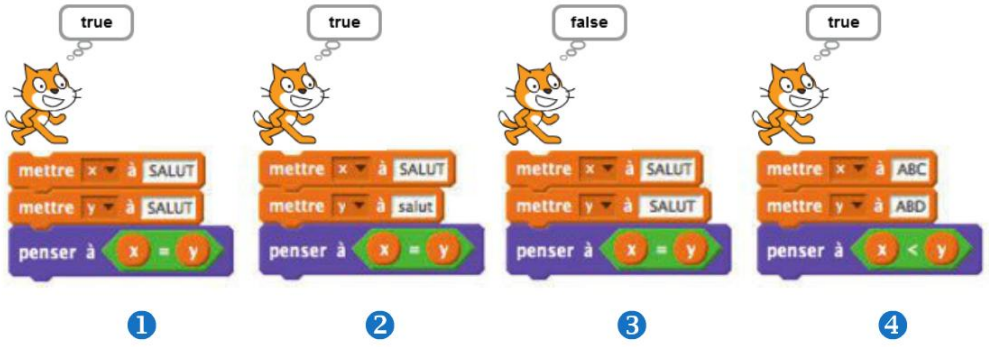


Figure 6-2 : L'utilisation d'opérateurs relationnels pour comparer ❶ des chaînes de caractères identiques, ❷ des chaînes qui ne diffèrent que par les majuscules, ❸ une chaîne avec une autre qui contient en plus des espaces et ❹ des chaînes qui diffèrent selon l'ordre alphabétique des lettres.

Copyright © 2016 Eyrolles.

Une étude approfondie de la figure 6-2 montre les points suivants :

- Scratch ne tient pas compte des majuscules lors de la comparaison de chaînes de caractères. Par exemple, « SALUT » et « salut » ❷ sont considérées égales.
- Scratch prend en compte les espaces dans ses comparaisons. La chaîne « SALUT », qui commence et termine par un espace, est différente de « SALUT » ❸.
- Lorsqu'on compare « ABC » et « ABD » ❹, Scratch s'intéresse d'abord au premier caractère de chaque chaîne. Puisqu'il est identique (la lettre *A* dans ce cas), Scratch passe au second caractère, également identique. le logiciel passe à la comparaison du troisième caractère. Puisque la lettre *C* est inférieure à la lettre *D* (*C* est placée avant *D* dans l'alphabet), Scratch considère que la première chaîne de caractères est plus petite que la seconde. Sachant cela, vous ne devriez pas être surpris en apprenant que l'expression **girafe > souris** est évaluée comme fausse, bien qu'une vraie girafe soit bien plus grosse qu'une vraie souris. D'après les règles de comparaison de Scratch, la chaîne de caractères « girafe » est inférieure à « souris » car la lettre *g* se situe avant *s* dans l'ordre alphabétique.
- La comparaison et le tri de chaînes de caractères sur la base de l'ordre alphabétique sont utiles dans beaucoup de situations courantes : ordre des noms dans un répertoire, des livres dans une étagère, des mots dans un dictionnaire, etc. Le mot *girafe* se trouve avant le mot *souris* dans un dictionnaire et la comparaison de chaînes de caractères dans Scratch donne une réponse basée sur cet ordre.

Maintenant que vous comprenez ce que sont les opérateurs relationnels et comment Scratch les utilise pour comparer des nombres et des chaînes de caractères, il est temps d'étudier les blocs conditionnels.

Les structures de décision

La palette *Contrôle* de Scratch contient deux blocs qui vous permettent de prendre des décisions et de contrôler les actions de vos programmes : les blocs **si** et **si/sinon**. En les utilisant, vous posez une question et choisissez une action en fonction de la réponse. Dans cette section, nous étudierons ces deux blocs en détail, parlerons des drapeaux et apprendrons à tester des conditions multiples avec les blocs **si** imbriqués. Je présenterai les applications pilotées par des menus et expliquerai la façon dont les blocs **si** peuvent servir pour leur implémentation.

Le bloc si

Le bloc **si** est une structure de décision qui vous permet de spécifier si un ensemble de commandes doit ou non être exécuté selon le résultat de l'évaluation d'une condition. La structure du bloc **si** et le diagramme sagittal correspondant sont représentés sur la figure 6-3.

Copyright © 2016 Eyrolles.

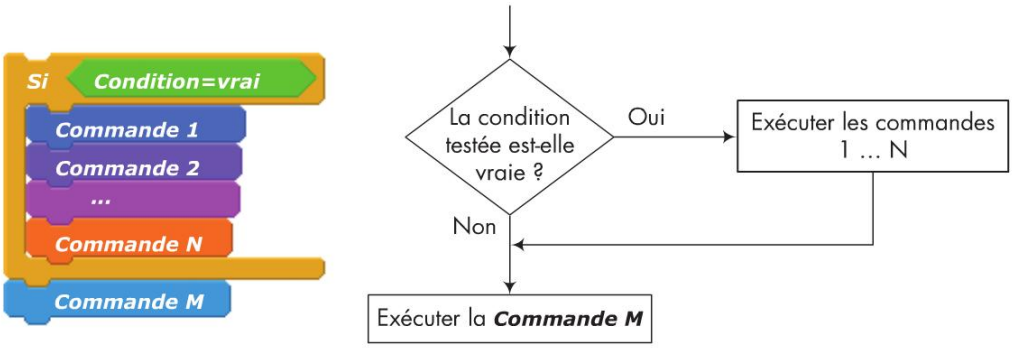


Figure 6-3 : Structure du bloc **si**

Dans la figure 6-3, le losange représente un bloc de décision qui donne une réponse oui/non (ou vrai/faux) à une question. Si la condition testée dans l'*en-tête* du bloc **si** est vraie, alors le programme exécute les commandes situées dans le *corps* avant de passer à la commande qui suit le bloc **si** (***Commande M*** sur la figure). Si la condition testée est fausse, le programme saute ces commandes et va directement à ***Commande M***.

Pour voir le bloc **si** en action, créez le script de la figure 6-4 et exécutez-le. Il exécute une boucle **répéter indéfiniment** qui déplace un lutin sur la scène, change sa couleur et le fait rebondir sur les bords.

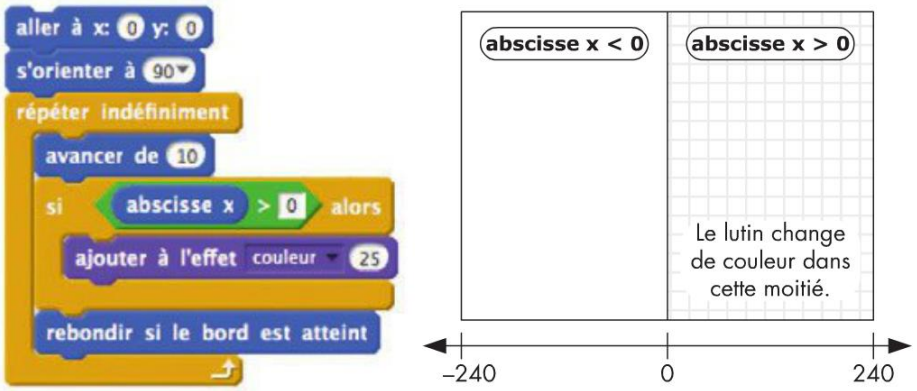


Figure 6-4 : Ce script change la couleur du lutin lorsqu'il se trouve dans la partie droite de la scène.

La boucle **répéter indéfiniment** de notre script contient un bloc **si** qui vérifie la position *x* du lutin après chaque commande **avancer**. Si cette position *x* est supérieure à zéro, le lutin change de couleur. Lorsque vous exécutez ce script, vous pouvez remarquer que le lutin ne change de couleur que lorsqu'il se déplace dans la partie droite de la scène. En effet, le bloc **ajouter à l'effet couleur 25** n'est exécuté que quand la condition **abscisse x > 0** est vraie.

Utiliser les variables comme drapeaux

Imaginons que nous développions un jeu d'aventure spatiale dont le but serait de détruire une flotte de vaisseaux de guerre assaillants. Le joueur, qui serait le capitaine, devrait manœuvrer un vaisseau spatial avec les flèches du clavier et tirer des missiles en appuyant sur la barre d'espace. Si le vaisseau du joueur se faisait toucher par un feu ennemi un certain nombre de

Copyright © 2016 Eyrolles.

fois, il perdrait sa capacité d'attaque. À ce stade, appuyer sur la barre d'espace ne lancerait plus de missile et le capitaine devrait adopter une stratégie de défense pour éviter de se faire toucher à nouveau. Il est clair que lorsque la barre d'espace sera pressée, votre programme devra vérifier l'état du système d'attaque du vaisseau spatial pour décider s'il peut tirer ou non.

Les vérifications de cette nature sont généralement exécutées grâce à des *drapeaux*. Ces derniers sont des variables utilisées pour indiquer si un événement a eu lieu on non. Vous pourriez utiliser n'importe quelle valeur pour décrire le statut de l'événement, mais la pratique usuelle préconise l'utilisation de 0 (ou faux) pour indiquer que l'événement n'a pas eu lieu et de 1 (ou vrai) pour indiquer qu'il a eu lieu.

Dans votre jeu de l'espace, vous utiliserez par exemple un drapeau nommé peutTirer pour indiquer l'état du vaisseau. Une valeur de 1 signifie que le vaisseau peut tirer un missile et une valeur de 0 signifie qu'il ne le peut pas. Votre script de gestion de l'événement « barre d'espace pressée » peut donc être codé comme sur la figure 6-5.

Au démarrage du jeu, vous initialisez la valeur du drapeau peutTirer à 1. Une fois le vaisseau touché par un certain nombre de tirs ennemis, vous réglez le drapeau peutTirer à 0 pour indiquer que le système d'attaque n'est plus fonctionnel ; à ce stade, une pression sur la barre d'espace ne lance plus de missile.

Bien que vous puissiez donner n'importe quel nom à vos drapeaux, je recommande des noms qui reflètent leur nature vrai/faux. Le tableau 6-3 montre quelques exemples que vous pourriez utiliser dans le jeu de l'espace.



Figure 6-5 : Utilisation d'un drapeau pour l'évaluation d'une condition

Tableau 6-3 : Quelques exemples d'utilisation des drapeaux

Exemple	Signification et action possible
mettre jeuDémarré à 0	Le jeu n'a pas encore commencé. Ignorer tout appui sur une touche du clavier.
mettre jeuDémarré à 1	Le jeu a commencé. Démarrer le traitement des entrées de l'utilisateur.
mettre jeuTerminé à 0	Le jeu n'est pas encore terminé. Montrer le temps restant.
mettre jeuTerminé à 1	Le jeu est terminé. Cacher l'affichage du temps restant.
mettre touché à 0	Le vaisseau n'est pas touché par un feu ennemi. Le son d'alarme est éteint.
mettre touché à 1	Le vaisseau a été touché par un missile. Jouer le son de l'alarme.

Maintenant que vous savez utiliser le bloc **si** et les drapeaux, abordons un autre bloc conditionnel, qui exécute un ensemble de commandes si une condition est vraie et un autre ensemble de commandes si elle est fausse.

Le bloc si/sinon

Imaginons que vous créiez un jeu qui apprenne les bases des mathématiques à des élèves de primaire. Il présenterait une addition et demanderait

Copyright © 2016 Eyrolles.

à l'élève de rentrer une réponse. L'élève recevrait un point pour chaque réponse correcte et perdrait un point pour chaque réponse incorrecte. Vous pouvez exécuter cette tâche en utilisant deux expressions **si** :

```
Si la réponse est correcte, ajouter un point à score.
Si la réponse est incorrecte, soustraire un point à score.
```

Vous pourriez simplifier cette logique – et rendre le code plus efficace – en combinant les deux expressions **si** dans une seule expression **si/sinon** :

```
Si la réponse est correcte
  ajouter un point à score.
Sinon
  soustraire un point à score.
```

La condition spécifiée est testée. Si elle est vraie, les commandes de la partie **si** du bloc sont exécutées. Si elle est fausse, ce sont les commandes de la partie **Sinon** qui s'exécutent. Le programme n'exécute que l'un des deux groupes de commandes du bloc **si/sinon**. Ces chemins alternatifs que peut prendre le programme sont appelés des *branches*. La structure du bloc **si/sinon** et son diagramme sagittal sont représentés à la figure 6-6.

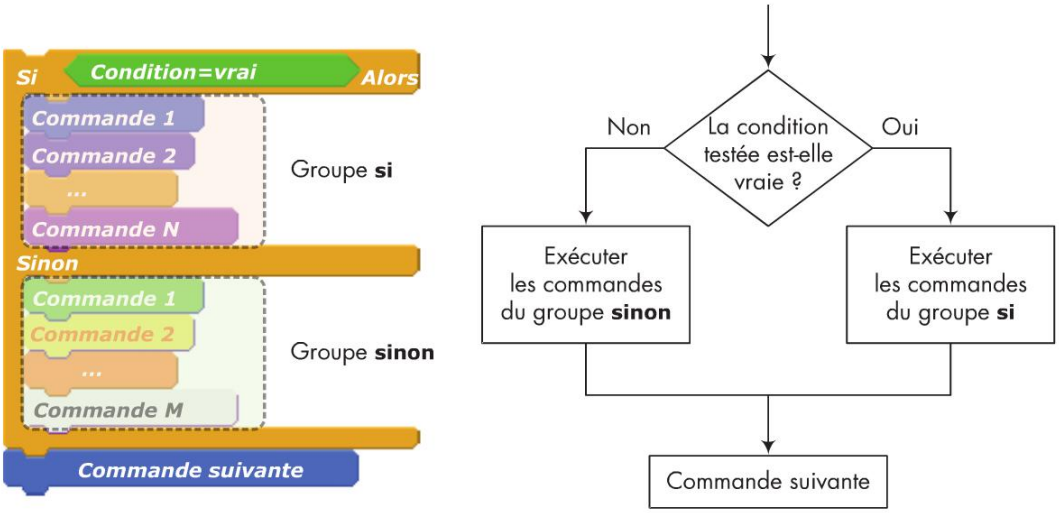


Figure 6-6 : Structure du bloc **si/sinon**

Dans la vraie vie, vous utilisez par exemple la structure **si/sinon** lorsque vous décidez où vous allez manger. Si vous avez assez d'argent, vous irez dans un restaurant branché ; sinon, vous vous dirigerez vers un restaurant moins huppé. Appelons l'argent qui se trouve dans votre porte-monnaie argentDisponible. Quand vous ouvrez votre porte-monnaie, vous vérifiez la condition **argentDisponible > 20** €. Si le résultat est vrai (si vous avez plus de 20 €), vous irez là où il y a des nappes, sinon vous irez au kébab du coin.

Un script simple qui illustre l'utilisation du bloc **si/sinon** est présenté à la figure 6-7. Il utilise l'*opérateur modulo* (**mod**), qui renvoie le reste d'une division, pour déterminer si un nombre entré par l'utilisateur est pair ou impair. Souvenez-vous qu'un nombre pair a un reste de zéro lorsqu'il est divisé par deux.

Copyright © 2016 Eyrolles.

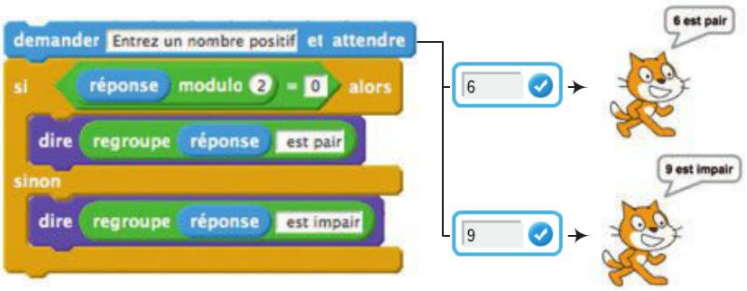


Figure 6-7 : Ce script détecte si un nombre entré est pair ou impair.

La figure 6-7 montre deux exemples de sorties lors desquels l'utilisateur entre 6 et 9, respectivement, en réponse à la commande **demander**. Pouvez-vous expliquer le fonctionnement de ce script ?

Blocs si et si/sinon imbriqués

Si vous voulez tester plus d'une condition avant d'effectuer une action, vous pouvez imbriquer des blocs **si** (ou **si/sinon**) pour effectuer tous les tests nécessaires. Considérez par exemple le script de la figure 6-8, qui détermine si un étudiant doit recevoir une bourse. Pour être éligible, l'étudiant doit avoir (1) une moyenne supérieure à 15/20 et (2) une note en maths supérieure à 18,5/20.

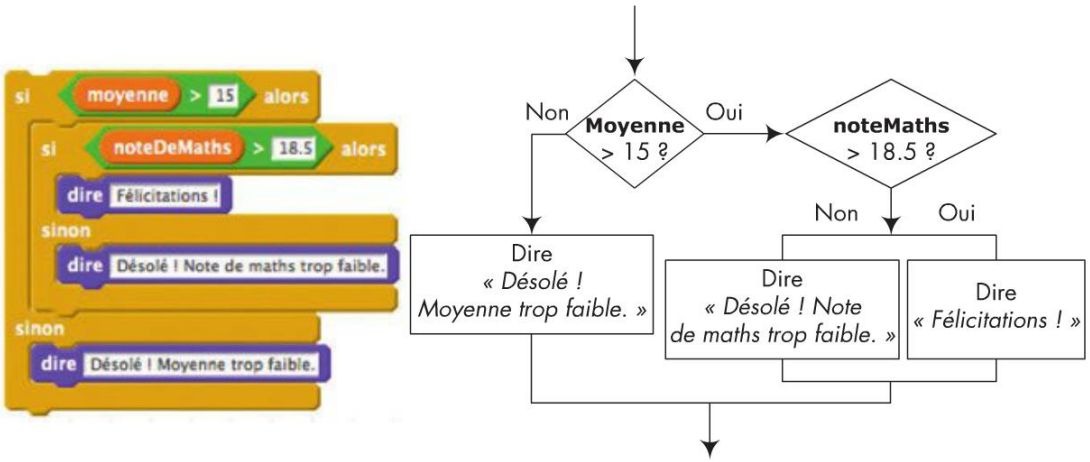


Figure 6-8 : Utilisation de blocs **si/sinon** *imbriqués pour tester plusieurs conditions*

Tout d'abord, l'expression **moyenne > 15** est évaluée. Si elle est fausse, il est inutile de tester l'autre condition puisque l'étudiant ne répond pas aux critères d'éligibilité. Si l'expression **moyenne > 15** est vraie, cependant, nous devrons tester la deuxième condition. Ceci est effectué par le bloc **si/sinon** imbriqué, qui teste la condition **noteMaths > 18.5**. Si la seconde condition est également vraie, l'étudiant obtient sa bourse. Sinon, il ne répond pas aux critères et un message approprié expliquant les raisons du refus est affiché.

Programmes pilotés par des menus

CalculateurdAire .sb2

Par la suite, nous allons étudier un cas typique d'utilisation de blocs **si** imbriqués. Vous apprendrez plus particulièrement à écrire des programmes qui présentent plusieurs choix à l'utilisateur et qui agissent en fonction de sa sélection.

Copyright © 2016 Eyrolles.

Lorsque vous les démarrez, certains programmes affichent une liste (ou un menu) des options disponibles et attendent que vous fassiez une sélection. Parfois, vous interagissez avec ces programmes en entrant un nombre correspondant à l'option choisie. De tels programmes peuvent utiliser des séquences de blocs **si/sinon** imbriqués pour déterminer la sélection de l'utilisateur et agir en conséquence. L'application présentée à la figure 6-9, qui calcule l'aire de différentes formes géométriques, montre comment fonctionnent les blocs **si/sinon** imbriqués.

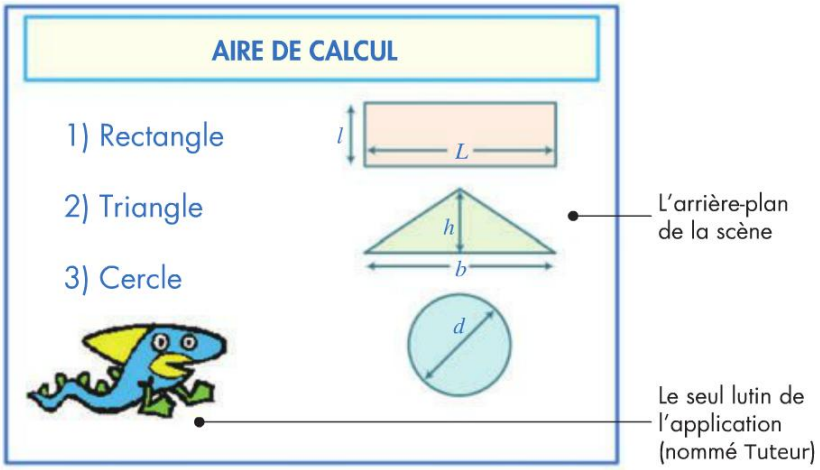


Figure 6-9 : L'interface utilisateur du programme de calcul d'aire

L'interface utilisateur de cette application contient l'arrière-plan de la scène, qui montre les options disponibles (les nombres 1, 2 et 3) et le lutin Tuteur, qui demande à l'utilisateur de faire un choix, effectue le calcul et affiche le résultat. Le script principal, représenté à la figure 6-10, démarre son exécution quand on clique sur le drapeau vert.

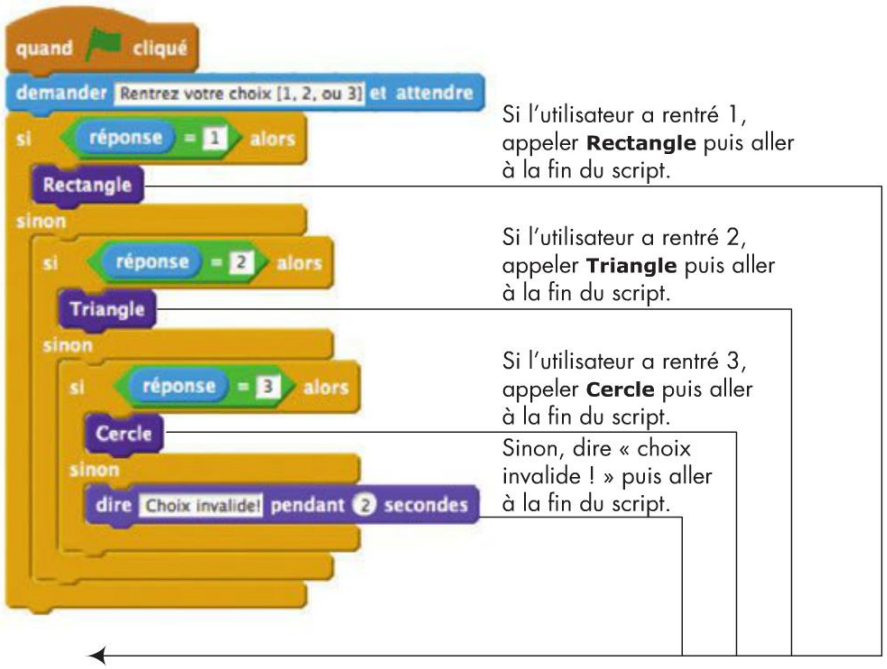


Figure 6-10 : Le script principal du lutin Tuteur

Copyright © 2016 Eyrolles.

Après avoir demandé à l'utilisateur d'entrer son choix, le lutin Tuteur attend sa réponse et utilise trois blocs **si/sinon** pour agir en conséquence. Si l'utilisateur entre un choix valide (1, 2 ou 3), le script appelle la procédure appropriée pour calculer l'aire de la forme choisie. Sinon, le script fait appel à la commande **dire** pour indiquer à l'utilisateur que le choix entré est invalide. Les procédures de calcul des aires des trois formes sont présentées sur la figure 6-11.

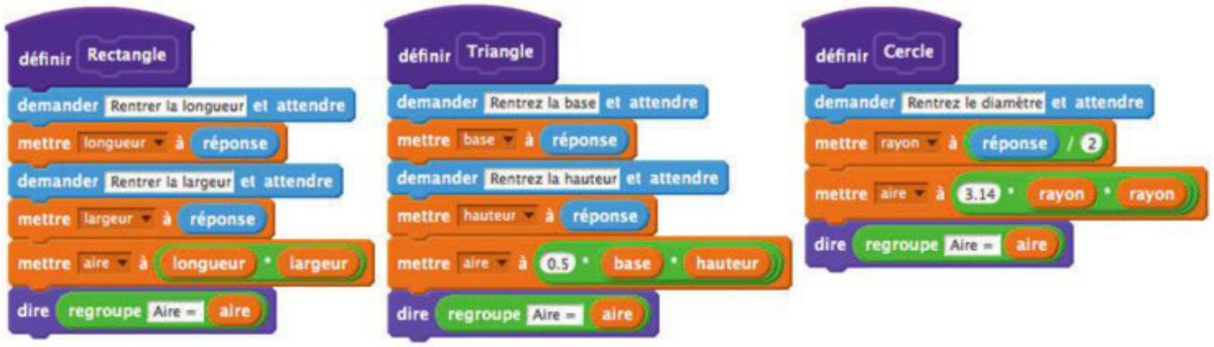


Figure 6-11 : Procédures du programme de calcul d'aire

Chaque procédure demande à l'utilisateur d'entrer les dimensions de la forme correspondante, calcule son aire et affiche le résultat. Par exemple, la procédure **Rectangle** demande à l'utilisateur d'entrer la longueur et la largeur du rectangle et enregistre les réponses dans les variables longueur et largeur, respectivement. Il calcule ensuite l'aire en multipliant ces deux variables et affiche la réponse. Les deux autres procédures fonctionnent de façon similaire.

Les opérateurs logiques

Dans la section précédente, vous avez appris à utiliser des blocs **si** et **si/alors** imbriqués pour tester des conditions multiples, mais vous pouvez également le faire avec les *opérateurs logiques*. Ces derniers combinent plusieurs expressions relationnelles pour produire un résultat vrai/faux unique. Par exemple, l'expression logique `(x > 5)` et `(x < 10)` est constituée des deux expressions (ou opérandes) `x > 5` et `x < 10` combinées grâce à l'opérateur logique **et** ; le résultat de cet opérateur ne sera vrai que si les deux opérandes sont vrais. Le tableau 6-4 liste les trois opérateurs logiques disponibles dans Scratch avec une brève explication de leur signification.

Tableau 6-4 : Les opérateurs logiques

Opérateur	Signification
	Le résultat n'est vrai que si les deux expressions sont vraies.
	Le résultat est vrai si l'une des deux expressions est vraie.
	Le résultat est vrai si l'expression est fausse.

Maintenant que vous avez vu un aperçu de chaque opérateur, étudions-les un par un plus en détail.

Copyright © 2016 Eyrolles.

L'opérateur et

L'opérateur **et** prend deux expressions comme paramètres d'entrée. Si les deux sont vraies, l'opérateur **et** renvoie vrai ; sinon, il renvoie faux. La table de vérité du **et**, qui fait la liste des sorties de l'opérateur pour toutes les combinaisons d'entrées possibles, est représentée dans le tableau 6-5.

Tableau 6-5 : Table de vérité de l'opérateur et

X	Y	X et Y
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Pour vous donner un exemple d'utilisation de l'opérateur **et**, imaginons que nous voulions créer un jeu dans lequel le joueur obtiendrait 200 points de bonus lorsque le score atteint 100 dans le premier niveau. La figure 6-12 montre la façon dont ces conditions peuvent être testées en utilisant des blocs **si** imbriqués ❶ ou l'opérateur **et** ❷.



Figure 6-12 : Vérification de conditions multiples avec l'utilisation de blocs **si** *imbriqués ou de l'opérateur* **et**.

Dans les deux cas, les points bonus ne sont ajoutés que quand les deux conditions sont vraies. Comme vous le constatez, l'opérateur **et** fournit une manière plus concise d'effectuer ce test. La commande à l'intérieur du bloc **si** de la figure 6-12 ❷ n'est exécutée que si niveau est égal à 1 et score est égal à 100. Si l'une des deux conditions est fausse, le test entier est évalué comme faux et le bloc **ajouter à score 200** n'est pas exécuté.

L'opérateur ou

L'opérateur **ou** prend également deux expressions comme paramètres d'entrée. Si l'une de ces expressions est vraie, il renvoie vrai. Il ne retourne faux que si les deux expressions d'entrée sont fausses. Sa table de vérité est présentée dans le tableau 6-6.

Copyright © 2016 Eyrolles.

Tableau 6-6 : La table de vérité de l'opérateur ou

X	Y	X ou Y
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Pour démontrer l'utilisation de l'opérateur **ou**, imaginons que les joueurs d'un certain jeu aient un temps limité pour atteindre le niveau suivant. Ils démarreraient également le jeu avec une quantité d'énergie qui diminuerait lors de leur déplacement dans le niveau actuel. Le jeu se terminerait si le joueur n'atteignait pas le niveau suivant dans le temps imparti ou s'il perdait toute son énergie avant de l'avoir atteint. La figure 6-13 montre comment la condition de fin du jeu pourrait être testée en utilisant des blocs **si/sinon** imbriqués ❶ ou l'opérateur **ou** ❷.

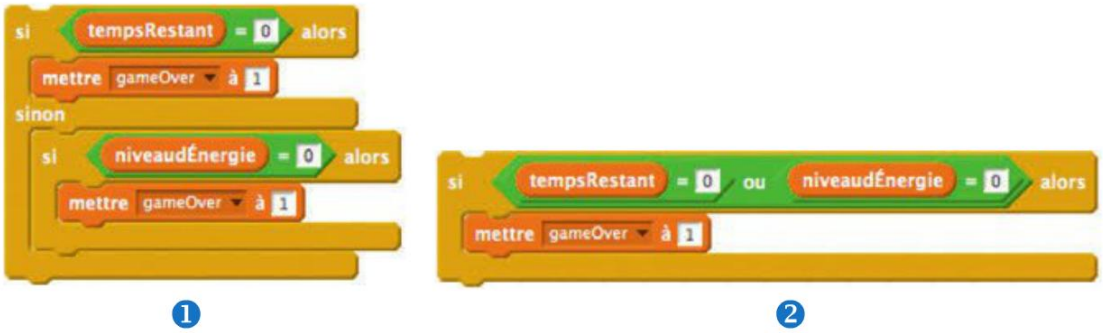


Figure 6-13 : Vérification de conditions multiples avec l'utilisation de blocs **si** *imbriqués ou de l'opérateur* **ou**.

Notez à nouveau que l'opérateur **ou** offre une manière plus concise de tester des conditions multiples. La commande à l'intérieur du bloc **si** de la figure 6-13 ▯ est exécutée si tempsRestant ou niveaudEnergie est égal à 0. Si les deux conditions sont fausses, le test entier est faux et le drapeau gameOver n'est pas mis à 1.

L'opérateur non

L'opérateur **non** ne prend qu'une seule expression comme paramètre d'entrée. Son résultat est vrai si l'expression d'entrée est fausse, et inversement. Sa table de vérité est représentée dans le tableau 6-7.

Tableau 6-7 : Table de vérité de l'opérateur non

X	non X
Vrai	Faux
Faux	Vrai

Copyright © 2016 Eyrolles.

En retournant à notre jeu hypothétique précédent, imaginons que le joueur ne puisse pas passer au niveau suivant si le score n'est pas supérieur à 100. L'opérateur **non** pourrait alors être utilisé comme le montre la figure 6-14. Vous pouvez lire ce code de la manière suivante : « si score n'est pas supérieur à 100, exécuter les commande à l'intérieur du bloc **si** ».



Figure 6-14 : Exemple d'utilisation de l'opérateur non

Si la valeur de score est inférieure ou égale à 100, le test est évalué comme vrai et la commande **dire** est exécutée. Notez que l'expression **non (score > 100)** est équivalente à **(score ≤ 100)**.

Utiliser les opérateurs logiques pour tester des intervalles

Lorsque vous devez valider des données entrées par l'utilisateur ou filtrer les entrées invalides, servez-vous des opérateurs logiques pour déterminer si un nombre est compris ou non dans un intervalle. Le tableau 6-8 montre quelques exemples d'intervalles.

Tableau 6-8 : Évaluer des intervalles

Expression	Valeur
(x > 10) et (x < 20)	Vrai si **x** est compris entre 10 et 20
(x < 10) ou (x > 20)	Vrai si **x** est inférieur à 10 ou supérieur à 20
(x < 10) et (x > 20)	Toujours faux : **x** ne peut pas être à la fois inférieur à 10 et supérieur à 20.

Bien que Scratch n'ait pas d'opérateur ≤ (inférieur ou égal) ou ≥ (supérieur ou égal) intégré, vous pouvez utiliser les opérateurs logiques pour implémenter ces tests. Par exemple, imaginons que vous vouliez tester la condition **x ≥ 10** dans votre programme. L'ensemble des solutions de cette inégalité est représenté à la figure 6-15 ❶. Un cercle plein sur la figure signifie que le nombre 10 est inclus dans l'ensemble des solutions.

Une première manière de tester cette condition est présentée à gauche de la figure 6-15 ❷. Cette dernière montre l'ensemble des solutions pour **x < 10**, où les cercles vides signifient que le point correspondant n'appartient pas à l'ensemble de solutions. Vous constatez que la solution *complémentaire* (« *x* n'est pas inférieur à 10 ») est équivalente à **x ≥ 10**. Une autre manière de tester l'inégalité est présentée à droite de la figure 6-15 ❸. De façon évidente, si **x ≥ 10**, alors *x* est supérieur à 10 ou égal à 10.

Copyright © 2016 Eyrolles.

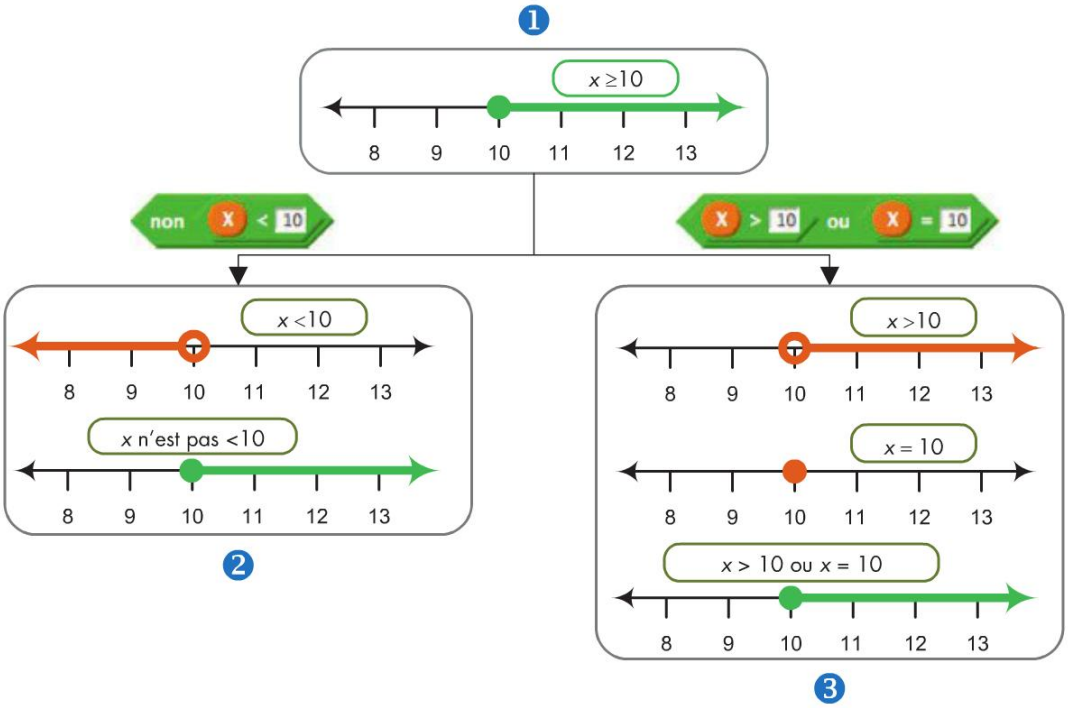


Figure 6-15 : Deux manières d'implémenter l'inégalité **x ≥ 10**

Les exemples fournis dans le tableau 6-9 montrent comment utiliser les opérateurs relationnels et logiques de Scratch pour exprimer des inégalités contenant les opérateurs ≤ et ≥.

Tableau 6-9 : Exemples de tests d'inégalités

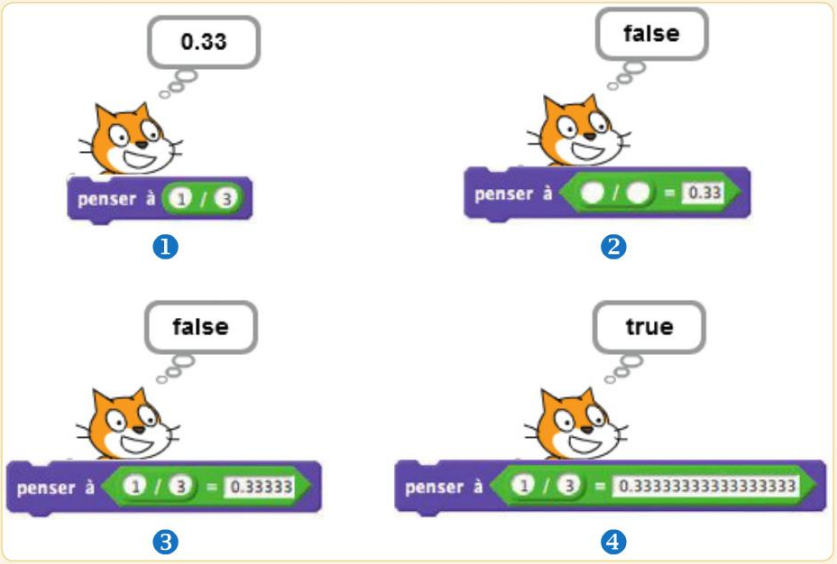
Expression	Implémentation utilisant des opérateurs logiques
x ≥ 10	non x < 10
x ≥ 10	x > 10 ou x = 10
x ≤ 10	non x > 10
x ≤ 10	x < 10 ou x = 10
10 ≤ x ≤ 20	non x < 10 et non x > 20
10 ≤ x ≤ 20	x > 10 et x < 20 ou x = 10 ou x = 20

À ce niveau du chapitre, nous avons étudié plusieurs concepts de Scratch, comprenant les comparaisons, les expressions conditionnelles et les opérateurs logiques. À présent, utilisons ces connaissances pour créer des applications amusantes et utiles.

Copyright © 2016 Eyrolles.

COMPARER DES NOMBRES DÉCIMAUX

Une attention particulière doit être portée lorsqu'on utilise l'opérateur égal pour comparer des nombres décimaux. À cause de la façon dont ces nombres sont stockés dans l'ordinateur, la comparaison est parfois imprécise. Considérez les blocs de commande suivants :



Le résultat de la division de 1 par 3 donne 0,3333… avec une infinité de 3. Puisque les ordinateurs utilisent un espace mémoire de taille finie pour stocker un résultat, la fraction 1/3 ne sera pas stockée de façon précise dans la mémoire. Bien que Scratch vous dise que le résultat de la division est 0,33 ❶, le vrai résultat est enregistré dans la mémoire avec une bien meilleure précision. Ainsi, les résultats des comparaisons ❷ et ❸ sont évalués comme faux.

Selon la situation de programmation, vous pourrez éviter ce type d'erreurs en utilisant l'une des approches suivantes :

- Lorsque cela est possible, préférez les opérateurs inférieur à (<) et supérieur à (>) à l'opérateur égal à (=).
- Utilisez le bloc **arrondi de** pour approcher les deux nombres que vous devez comparer et voir si les deux arrondis sont égaux.
- Testez la différence absolue des deux valeurs que vous voulez comparer. Par exemple, au lieu de tester si x est égal à y, il vaut mieux vérifier si la valeur absolue de la différence entre x et y est inférieure à un seuil de tolérance :



Selon la précision des nombres et la méthode pour les calculer, cette manière de faire peut être suffisante.

Copyright © 2016 Eyrolles.

Projets Scratch

Les nouvelles commandes que vous avez apprises dans ce chapitre devraient vous permettre de créer un grand panel d'applications Scratch utiles et j'espère que les projets de cette section vous inspireront. Je vous encourage à tester ces applications, comprendre commente elles fonctionnent et essayer de trouver comment les améliorer.

Devine mes coordonnées

DevineMes Coordonnées .sb2

Dans cette section, nous allons développer un jeu interactif qui teste les connaissances du joueur sur le système de coordonnées cartésien. Ce jeu contient un seul lutin (appelé Étoile) qui représente un point aléatoire de la scène (figure 6-16).

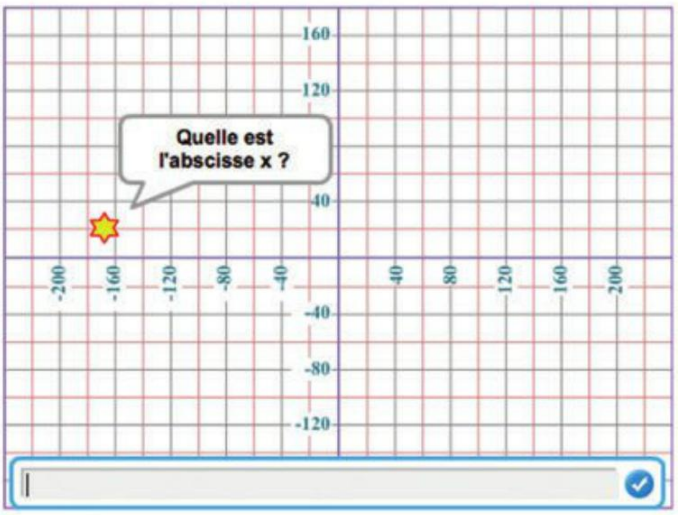


Figure 6-16 : L'interface de Devine mes coordonnées

À chaque fois que vous exécutez le jeu, le lutin se déplace à un endroit différent de la scène et demande au joueur de deviner ses coordonnées x et y. Le jeu vérifie la réponse du joueur et fournit un message de retour approprié. Le script principal du lutin Étoile est présenté à la figure 6-17.

Ce script utilise deux variables `X` et `Y` pour stocker les coordonnées aléatoires du lutin. Le fonctionnement de chaque section numérique de la figure 6-17 est expliqué ci-après.

1. La variable `X` se voit affecter une valeur aléatoire comprise dans l'ensemble {-220, -200, -180 … 220} : on choisit un entier aléatoire entre -11 et 11 et on multiplie le résultat par 20. De même, la variable `Y` prend une valeur aléatoire comprise dans l'ensemble {-160, -140, -120 … 160}. Les valeurs sélectionnées de `X` et `Y` assurent que le point correspondant soit situé sur l'une des intersections de la grille de la figure 6-16. Le lutin est ensuite déplacé à l'emplacement spécifié par `X` et `Y`.

Copyright © 2016 Eyrolles.

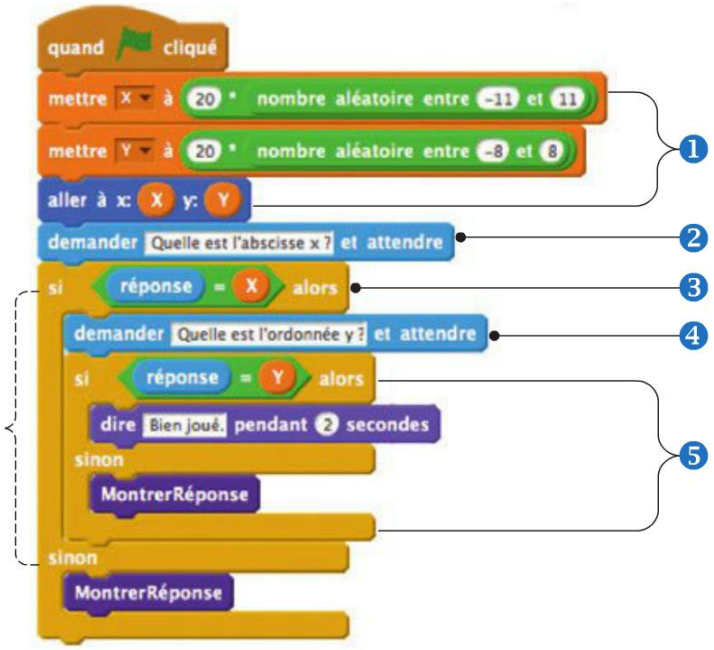


Figure 6-17 : Le script du jeu Devine mes coordonnées

2. Le script demande à l'utilisateur d'entrer son estimation de la coordonnée *x* du lutin et attend la réponse.
3. Si la réponse est correcte, le script passe à l'étape 4. Sinon, il appelle la procédure **MontrerRéponse** pour afficher les coordonnées correctes du point.
4. Lorsque l'utilisateur entre la valeur correcte pour *x*, le script lui demande d'entrer son estimation de la coordonnée *y* du lutin et attend la réponse.
5. Si la réponse est correcte, le script affiche le message « Bien joué ». Sinon, il appelle la procédure **MontrerRéponse** pour afficher les coordonnées correctes.

La procédure **MontrerRéponse** est représentée à la figure 6-18. Tout d'abord, la variable point est réglée de manière à contenir une chaîne de caractères de la forme (X,Y) en utilisant l'opérateur **regroupe**. La procédure utilise ensuite la commande **dire** pour montrer la réponse à l'utilisateur.



Figure 6-18 : La procédure **MontrerRéponse**

Copyright © 2016 Eyrolles.

FAITES LE TEST 6-1

Améliorez ce jeu de devinette en y ajoutant des modifications amusantes. Par exemple, vous pourriez faire en sorte que le jeu joue de la musique quand le joueur gagne, déclenche un buzz lors d'une fausse réponse, s'exécute automatiquement (sans avoir à cliquer sur le drapeau vert à chaque fois), ou enregistre le nombre de bonnes réponses pour afficher le score du joueur.

Jeu de classification de triangles

Classification DeTriangles.sb2

Comme le montre la figure 6-19, un triangle peut être classé comme scalène, isocèle ou équilatéral, selon la longueur de ses côtés. Dans cette section, nous allons étudier un jeu qui teste les joueurs sur ces concepts.

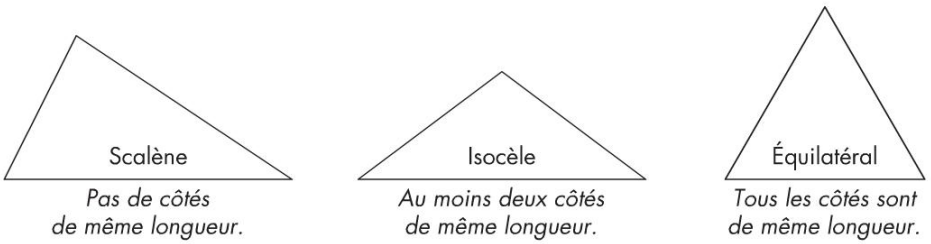


Figure 6-19 : Classification d'un triangle selon la longueur de ses côtés

Le jeu dessine un triangle sur la scène et demande au joueur de le classer dans l'un des trois types. L'interface utilisateur de ce jeu est présentée à la figure 6-20.

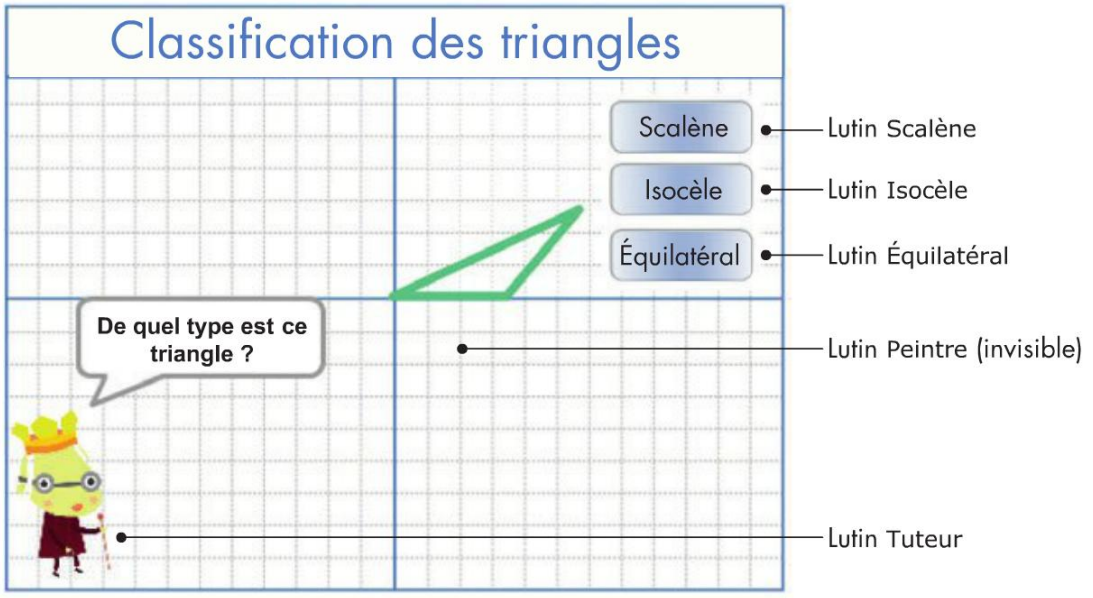


Figure 6-20 : L'interface utilisateur du jeu de classification de triangles

Copyright © 2016 Eyrolles.

La figure 6-20 montre que ce jeu contient cinq lutins. Trois d'entre eux (nommés Scalène, Isocèle et Équilatéral) représentent les boutons sur lesquels le joueur clique pour sélectionner une réponse. Le lutin invisible Peintre dessine le triangle sur la scène.

NOTE *J'ai rendu* Peintre *invisible en décochant sa case Montrer dans la zone d'information du lutin. Si vous préférez contrôler sa visibilité à partir du script, vous pouvez ajouter un bloc* **cacher** *pour le cacher de façon explicite au démarrage du jeu.*

Le lutin Tuteur est celui qui gère le jeu ; il détermine le type de triangle à dessiner à chaque exécution et vérifie la réponse du joueur. Ses scripts sont présentés à la figure 6-21.

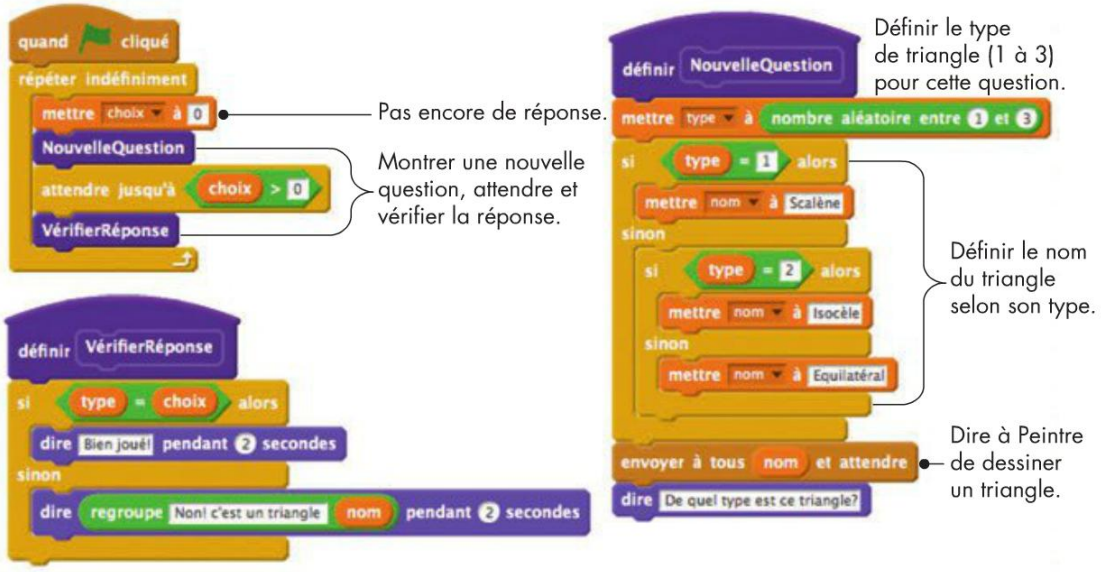


Figure 6-21 : Les scripts du lutin Tuteur. *Le script principal (en haut à gauche) appelle* **NouvelleQuestion** *(à droite) et* **VérifierRéponse** *(en bas à gauche).*

Lorsqu'on clique sur le drapeau vert pour démarrer le jeu, le script principal entre dans une boucle infinie. À chaque passage dans la boucle, le script met choix à 0 (pour indiquer que l'utilisateur n'a pas encore répondu), dessine un nouveau triangle et attend une réponse. La variable choix devrait changer lorsque l'utilisateur clique sur l'un des trois boutons. Ceci fait, le script vérifie la réponse et fournit un message de retour approprié. Regardons chaque étape en détail.

La procédure **NouvelleQuestion** démarre en réglant type de façon aléatoire – qui détermine le type du triangle dessiné sur la scène – à 1, 2 ou 3. Le script utilise ensuite un bloc **si/sinon** pour régler la variable nom selon la valeur de type. La variable nom a deux utilités : (1) elle indique quel message envoyer pour que le lutin Peintre sache quoi dessiner (observez la façon dont le bloc **envoyer à tous et attendre** utilise nom) et (2) elle est utilisée dans la procédure **VérifierRéponse** pour créer un message de retour approprié. Lorsque le lutin Peintre finit son dessin, la procédure **NouvelleQuestion** demande à l'utilisateur d'entrer une réponse grâce à la commande **dire**.

Copyright © 2016 Eyrolles.

Lorsque le lutin Peintre reçoit le message de Tuteur, il dessine le triangle correspondant sur la scène. Pour rendre le jeu plus amusant, le lutin Peintre utilise des valeurs aléatoires pour la taille, l'orientation et la couleur des triangles (figure 6-22).

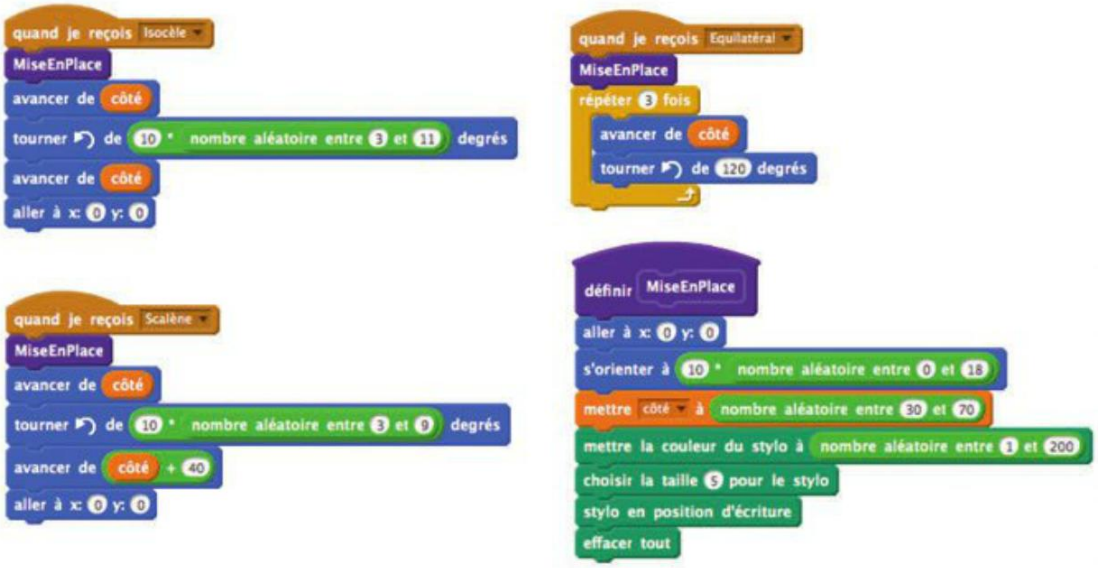
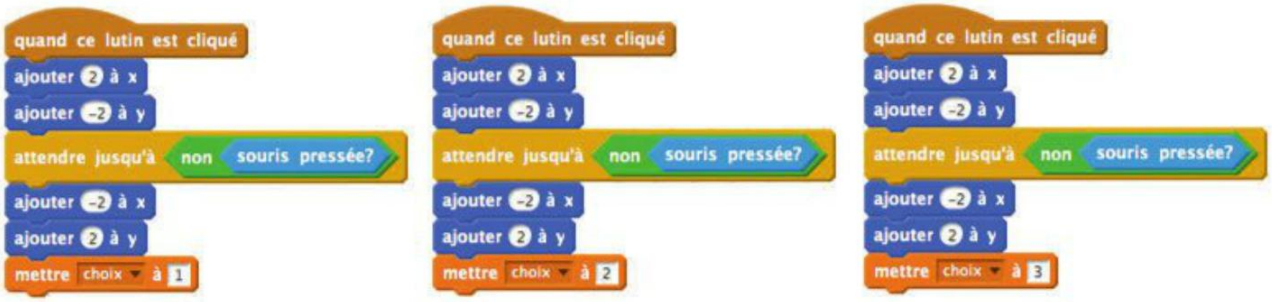


Figure 6-22 : Scripts du lutin Peintre

Après avoir demandé au joueur de classer le triangle dessiné, le script principal utilise le bloc **attendre jusqu'à** (de la palette *Contrôle*) et se met donc en pause jusqu'à ce que **choix > 0** soit vrai. Les trois lutins boutons changent la valeur de choix lorsqu'on clique dessus : le bouton Scalène mettra choix à 1, Isocèle à 2 et Équilatéral à 3 (figure 6-23).



Script du lutin Scalène | Script du lutin Isocèle | Script du lutin Équilatéral

Figure 6-23 : Les scripts des trois lutins boutons

Lorsque l'utilisateur clique sur un bouton, le lutin correspondant se déplace légèrement vers le bas et vers la droite pour donner l'impression d'un véritable appui. Lorsque la souris est relâchée, le lutin retourne à sa position initiale et règle la valeur de la variable choix. Chaque lutin bouton règle choix différemment. Les blocs qui déplacent les boutons dans ces scripts ne sont pas strictement nécessaires et peuvent éventuellement être supprimés.

Dès que l'utilisateur choisit le type de triangle, choix devient supérieur à zéro et le script principal appelle la procédure **VérifierRéponse**. Cette procédure compare la variable type (qui indique le type de triangle dessiné) à la variable

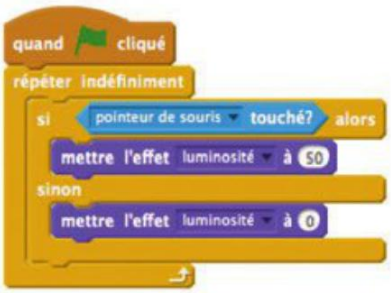
Copyright © 2016 Eyrolles.

choix. Si les deux variables ont la même valeur, alors la réponse du joueur est correcte. Sinon, la réponse du joueur est fausse et le script lui indique la bonne réponse.

FAITES LE TEST 6-2

Ouvrez ce jeu et jouez-y quelques fois. Une fois que vous aurez compris son fonctionnement, essayez d'y ajouter de nouvelles fonctions. Voici quelques pistes :

- Faire en sorte que le jeu enregistre le score. Il pourrait ajouter un point à chaque bonne réponse et en enlever un à chaque mauvaise réponse.
- Fournir une option permettant de quitter le jeu.
- Définir un critère de fin de jeu. Par exemple, vous pourriez faire en sorte que la boucle principale ne s'exécute que 20 fois au lieu de s'exécuter indéfiniment. Vous pourriez aussi arrêter le jeu au bout de cinq mauvaises réponses.
- Faire apparaître quelque chose de spécial à certains moments du jeu. Par exemple, vous pourriez créer une variable nommée nombreSpécial et lui donner une valeur aléatoire au début du jeu. Lorsque le nombre de bonnes réponses atteint la valeur de nombreSpécial, le jeu pourrait donner des points bonus au joueur, jouer de la musique, ou même dire une blague.
- Donner vie aux boutons avec des effets graphiques. Par exemple, vous pourriez ajouter le script présenté ci-après à tous les boutons pour qu'ils changent d'apparence lorsque la souris les survole.



Suiveur de ligne

SuiveurDeLigne .sb2

Pouvons-nous faire en sorte qu'un lutin suive en autonomie un chemin tracé sur la scène, comme celui de la figure 6-24 ? La réponse est oui et nous écrirons le programme correspondant dans cette section. En observant le zoom sur la tête du chat sur la figure, vous remarquerez que nous avons peint son nez en rose et ses deux oreilles avec des couleurs différentes.

Copyright © 2016 Eyrolles.

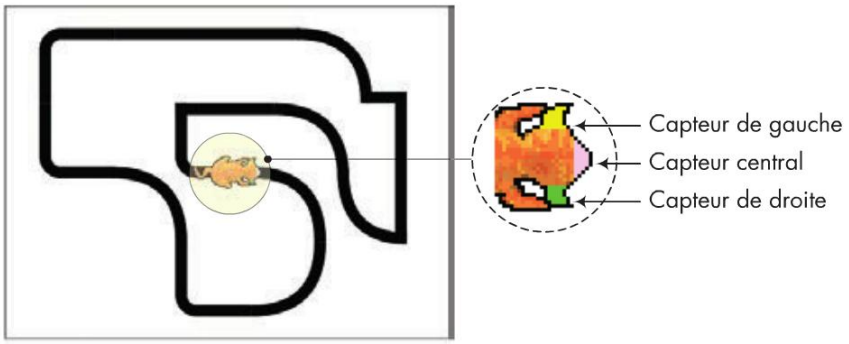


Figure 6-24 : Exemple de chemin que le lutin doit suivre

Nous allons utiliser le nez et les oreilles du chat comme des capteurs pour détecter la ligne noire que nous voulons lui faire suivre. Notre algorithme utilise les *heuristiques* suivantes (des règles basées essentiellement sur le raisonnement logique et sur l'expérimentation par essais successifs) :

- Si le nez du chat (le rose) touche la ligne, avancer tout droit.
- Si l'oreille gauche du chat (le jaune) touche la ligne, tourner dans le sens inverse des aiguilles d'une montre et avancer à allure réduite.
- Si l'oreille droite du chat (le vert) touche la ligne, tourner dans le sens des aiguilles d'une montre et avancer à allure réduite.

Bien sûr, la vitesse exacte (nombre de pas de déplacement) et les angles de rotation peuvent être différents pour différentes routes et doivent être déterminés par expérimentation.

Le script de la figure 6-25 implémente cet algorithme. Il utilise un nouveau bloc : **couleur touche ?** (de la palette *Capteurs*). Ce bloc vérifie si une couleur du lutin (spécifiée dans le premier carré coloré) touche une autre couleur (spécifiée dans le second carré). Si c'est le cas, le bloc retourne vrai ; sinon il retourne faux. La couleur d'un carré coloré peut être choisie en cliquant sur ce carré puis sur la couleur désirée n'importe où dans le projet Scratch.

Copyright © 2016 Eyrolles.

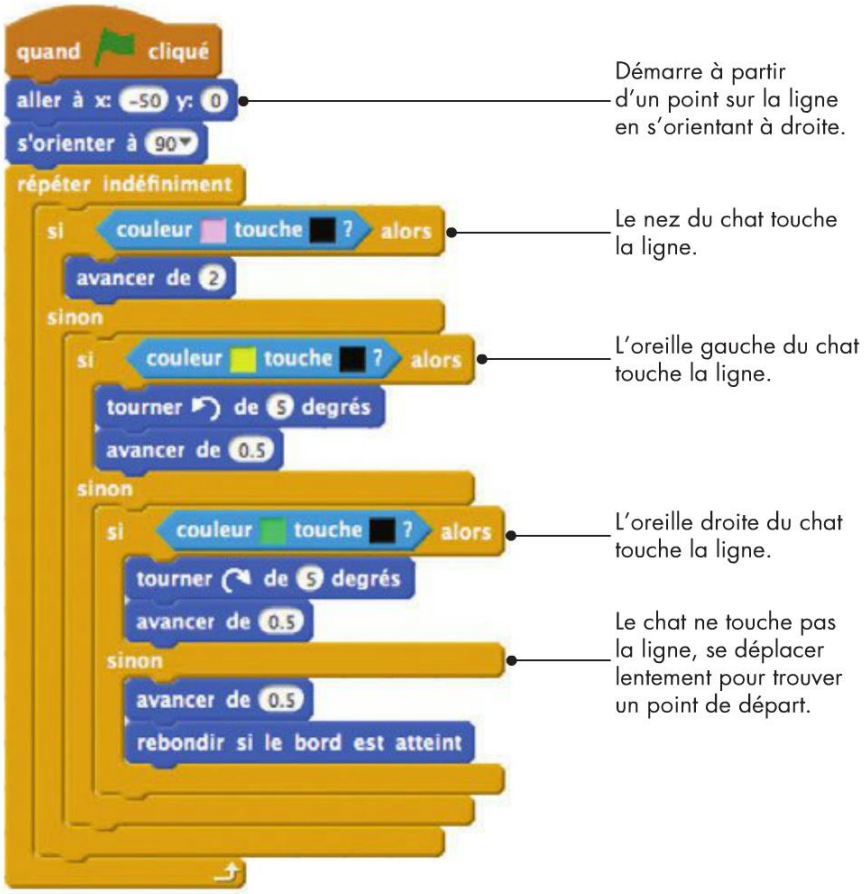


Figure 6-25 : L'algorithme du suiveur de ligne

FAITES LE TEST 6-3

Ouvrez l'application et exécutez-la pour voir comment elle fonctionne. Faites en sorte que le chat fasse le tour de la piste le plus rapidement possible en agissant sur les différentes valeurs du script. Un relecteur a réussi à le faire en 11 s. Pouvez-vous battre ce record ? Créez de nouvelles pistes et voyez si cet algorithme simple fonctionne toujours.

Équation d'une droite

ÉquationdUne Droite.sb2

L'équation d'une droite joignant deux points $P = (x_1, y_1)$ et $Q = (x_2, y_2)$ est $y = ax + b$, avec $a = (y_2 - y_1) / (x_2 - x_1)$ représentant la pente de la droite et b l'ordonnée à l'origine. Une droite verticale a une équation de la forme $x = k$ et une droite horizontale a une équation de la forme $y = k$, où k est une constante. Dans cette section, nous allons développer une application qui trouve l'équation de la droite liant deux points dans le plan cartésien. L'interface utilisateur de cette application est représentée à la figure 6-26.

Copyright © 2016 Eyrolles.

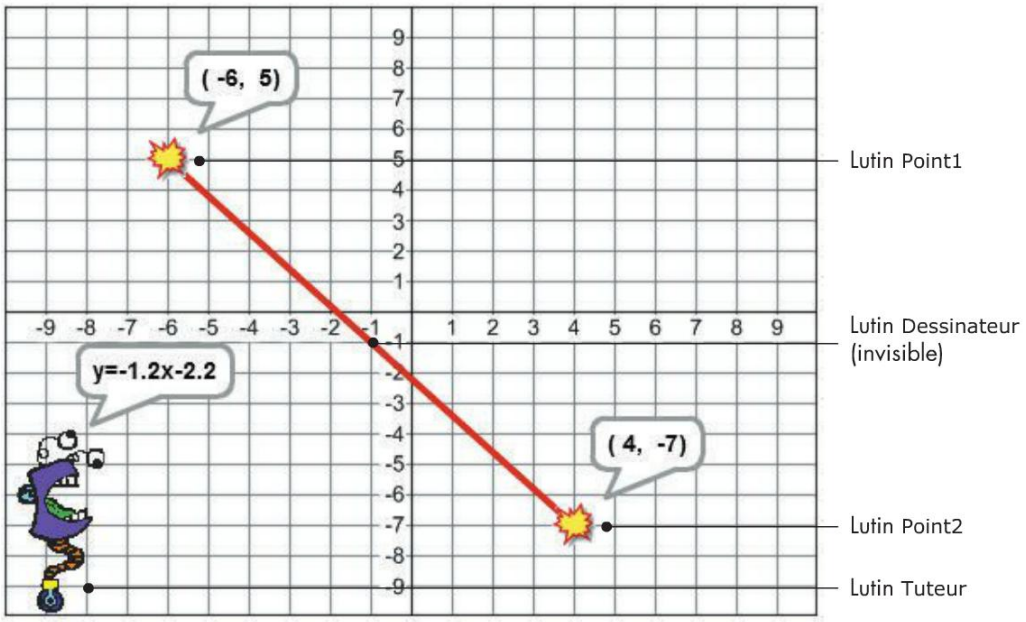


Figure 6-26 : L'interface utilisateur de l'application d'estimation d'équation

L'utilisateur fait glisser les deux lutins représentant les extrémités d'un segment sur la scène et l'application affiche automatiquement l'équation de la droite correspondante. Cette application contient quatre lutins : Point 1 et Point 2 sont utilisés pour définir les extrémités du segment, Dessinateur est utilisé pour tracer le segment liant ces deux points et Tuteur a pour rôle de calculer et d'afficher l'équation de la droite.

Les scripts de Point 1 et Point 2 sont très similaires. Ils contiennent des opérations logiques (qui ne sont pas montrées ici) qui restreignent leurs déplacements aux seuls points d'intersection de la grille. Le rôle principal du script de Point 1 est de mettre à jour les variables X1 etY1 qui contiennent ses coordonnées lorsque l'utilisateur le fait glisser, puis d'envoyer un message Redessiner. Il en est de même pour Point 2, si ce n'est que ses coordonnées sont appelées X2 et Y2. Les quatre variables (X1, X2, Y1 et Y2) ne peuvent prendre que des valeurs allant de -9 à 9. Vous trouverez le détail de ces scripts dans le fichier *EquationdUneDroite.sb2*. Jetons maintenant un œil aux scripts du lutin Dessinateur (figure 6-27).

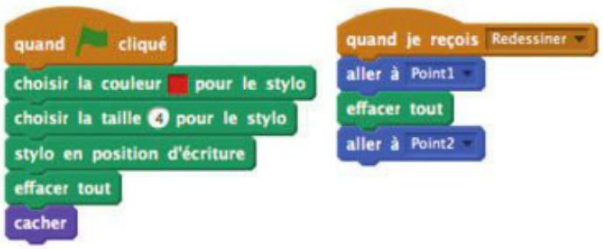


Figure 6-27 : Scripts du lutin Dessinateur

Copyright © 2016 Eyrolles.

Lorsque le jeu démarre, le lutin initialise la taille et la couleur de son stylo puis se prépare à dessiner. Lorsqu'il reçoit le message Redessiner, il se déplace au lutin Point 1, efface la scène et se déplace vers le lutin Point 2. Cela donne un segment qui connecte Point 1 et Point 2.

Le lutin Tuteur exécute également un script lorsqu'il reçoit le message Redessiner (figure 6-28).

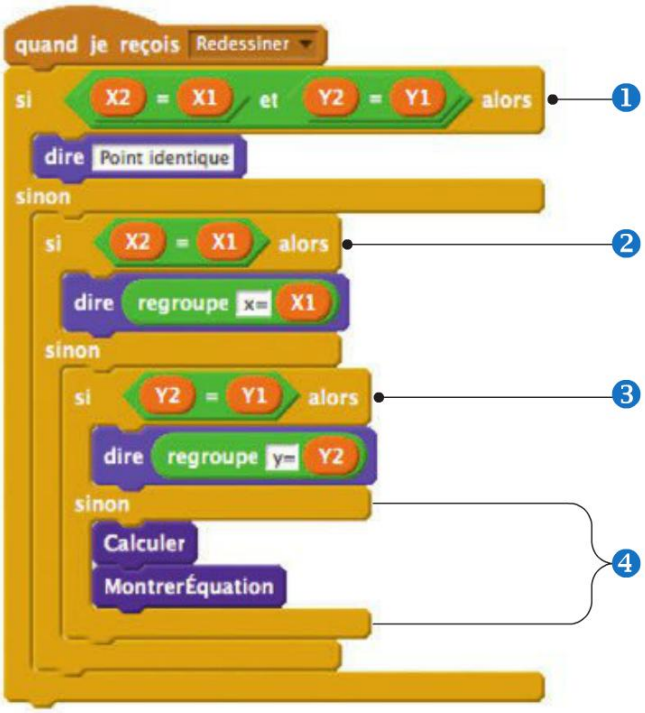


Figure 6-28 : Le script de gestion du message Redessiner *pour le lutin* Tuteur

Ce script effectue les vérifications suivantes :

- Si les coordonnées de Point 1 et de Point 2 sont les mêmes, il n'y a pas de droite. Le script indique donc simplement « Point identique ».
- Si les deux points sont différents mais partagent la même coordonnée x, la droite est verticale. Le script affiche alors une équation du type $x = constante$.
- Si les deux points sont différents mais partagent la même coordonnée y, la droite est horizontale. Le script affiche alors une équation du type $y = constante$.
- Sinon, les deux points appartiennent à une droite dont l'équation est de la forme $y = ax + b$. Le script appelle d'abord la procédure **Calculer** pour déterminer les valeurs des coefficients. Il appelle ensuite **MontrerÉquation** pour mettre l'équation dans le bon format et la montrer à l'utilisateur.

La procédure **Calculer** est présentée à la figure 6-29. Elle calcule la pente (a) et l'ordonnée à l'origine (b) puis arrondit ces valeurs au centième près.

Copyright © 2016 Eyrolles.

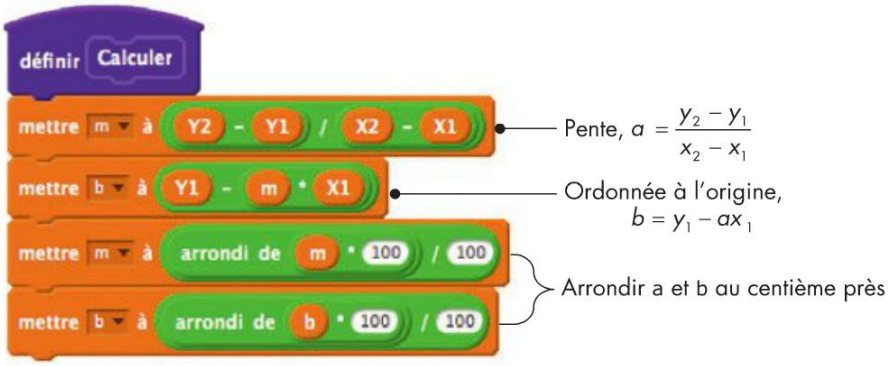


Figure 6-29 : La procédure **Calculer**

La procédure **MontrerÉquation** est représentée à la figure 6-30. Elle utilise deux variables (terme1 et terme2) et deux sous-procédures pour mettre l'équation à un format adapté à l'affichage.

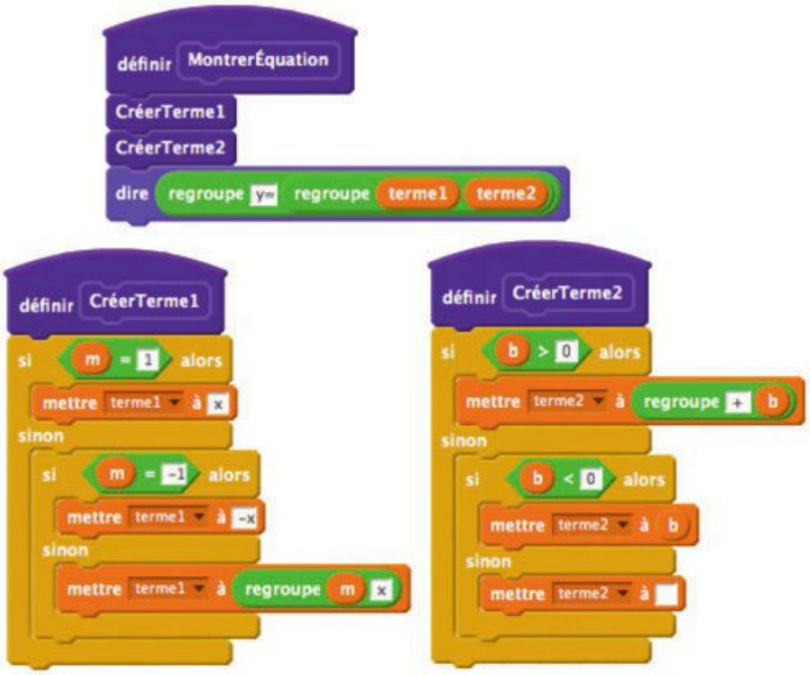


Figure 6-30 : La procédure **MontrerÉquation**

La procédure **MontrerÉquation** prend en compte les cas spéciaux suivants pour mettre l'équation au bon format :

- Si la pente est de 1, alors terme1 sera x (au lieu de $1x$).
- Si la pente est de -1, alors terme1 sera $-x$ (au lieu de $-1x$).
- terme2 est mis en forme en utilisant le signe (plus ou moins) approprié pour l'ordonnée à l'origine.
- Si l'ordonnée à l'origine est de 0, alors l'équation prendra la forme $y = mx$.

Copyright © 2016 Eyrolles.

FAITES LE TEST 6-4

Ouvrez l'application et exécutez-la. Fates glisser les deux points à des endroits différents sur la scène et vérifiez l'équation affichée. Pour améliorer cette application, essayez d'y ajouter un script qui déplace le lutin Tuteur s'il chevauche les lutins Point 1 ou Point 2.

Autres applications

GuessMyNumber.sb2

Voyons maintenant quelques jeux que vous trouverez dans les applications bonus de ce livre (à télécharger à partir de *www.editions-eyrolles.com/go/grandlivrescratch*). Ce dossier contient deux jeux classiques que vous pourrez explorer par vous-même. Le premier est *Guess My Number* (« Devine mon numéro » en français). Cette application sélectionne secrètement un nombre aléatoire entre 1 et 100 et demande au joueur de deviner quel numéro a été choisi. Elle indique ensuite au joueur si sa réponse est trop grande ou trop petite. Le joueur gagne s'il trouve la bonne réponse et perd au bout de la sixième mauvaise réponse.

RockPaper.sb2

Le deuxième jeu permet à l'utilisateur de jouer à *RockPaper* (Pierre-feuille-ciseaux en français) contre l'ordinateur. Le joueur choisit ce qu'il veut jouer en cliquant sur l'un des trois boutons représentant les trois choix possibles, puis l'ordinateur fait à son tour un choix aléatoire parmi ces possibilités. Le gagnant est sélectionné selon les règles suivantes : le papier bat (enveloppe) la pierre, la pierre bat (casse) les ciseaux, les ciseaux battent (coupent) la feuille.

Résumé

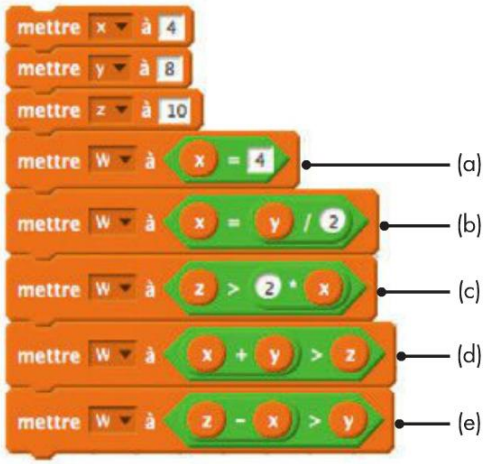
Dans ce chapitre, vous avez pris connaissance des opérateurs de comparaison de Scratch et avez appris à vous en servir pour comparer des nombres, des caractères ou des chaînes de caractères. Ensuite, vous avez découvert les blocs **si** et **si/sinon** et les avez utilisés pour prendre des décisions et contrôler des actions dans plusieurs programmes. Vous avez appris à utiliser des blocs **si** et **si/sinon** imbriqués pour tester des conditions multiples et vous avez appliqué cette technique pour développer des applications pilotées par des menus. Vous avez également appris à vous servir des opérateurs logiques comme un moyen alternatif et plus concis de tester des conditions multiples. Enfin, vous avez étudié plusieurs applications complètes qui ont illustré comment utiliser des structures de prise de décision.

Le prochain chapitre vous plongera dans la palette *Contrôle*, en vous montrant différentes structures de répétition disponibles dans Scratch et en vous enseignant comment les utiliser pour écrire des programmes encore plus puissants.

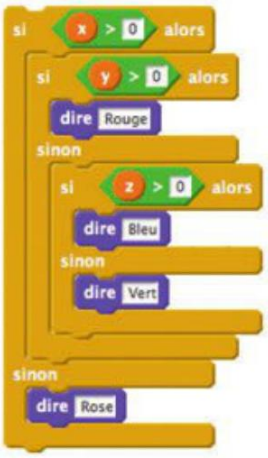
Copyright © 2016 Eyrolles.

Problèmes

1. Quelle sera la valeur de W après l'exécution du script suivant ?



2. Exprimez chacune des déclarations suivantes en utilisant un bloc **si** :
   - Si x divisé par y donne 5, alors mettre x à 100.
   - Si x fois y donne 5, alors mettre x à 1.
   - Si x est inférieur à y, alors doubler x.
   - Si x est supérieur à y, alors incrémenter x de 1.
3. Écrivez un programme qui demande à l'utilisateur de rentrer cinq notes comprises entre 1 et 10. Le programme comptera ensuite le nombre de notes supérieures à 7.
4. Exprimez chacune des déclarations suivantes en utilisant un bloc **si/sinon** :
   - Si x fois y donne 8, alors mettre x à 1 ; sinon, mettre x à 2.
   - Si x est inférieur à y, alors doubler x ; sinon, incrémenter x de 1.
   - Si x est inférieur à y, alors incrémenter x et y de 1 ; sinon, décrémenter x et y de 1.
5. Pour chaque déclaration suivante, estimez quelle sera la sortie du script :
   - x = -1, y = -1, z = -1
   - x = 1, y = 1, z = 0
   - x = 1, y = -1, z = 1
   - x = 1, y = -1, z = -1



Copyright © 2016 Eyrolles.

6. Écrivez un programme qui demande à l'utilisateur d'entrer trois nombres, détermine ensuite quel est le plus grand des trois nombres et l'affiche.
7. Une entreprise vend cinq produits différents dont les prix au détail sont présentés dans le tableau suivant. Écrivez un programme qui demande à l'utilisateur de rentrer le numéro du produit et la quantité vendue. Le programme calcule et affiche ensuite le revenu total correspondant.

Numéro de produit	Prix au détail
1	12,95 €
2	4,99 €
3	5,49 €
4	7,80 €
5	8,85 €

8. Construisez une expression logique pour représenter chacune des conditions suivantes :
   - score est supérieur à 90 et inférieur à 95.
   - réponse est soit o soit oui.
   - réponse est un nombre pair compris entre 1 et 10.
   - réponse est un nombre impair compris entre 1 et 10.
   - réponse est un nombre compris entre 1 et 5 mais pas égal à 4.
   - réponse est un nombre compris entre 1 et 100 et divisible par 3.
9. Le théorème de *l'inégalité triangulaire* affirme que la somme des longueurs de deux côtés est supérieure à la longueur du troisième côté. Écrivez un programme qui demande à l'utilisateur de rentrer trois nombres et qui lui indique si ces nombres peuvent ou non représenter les côtés d'un triangle.
10. Le *théorème de Pythagore* affirme que si a et b sont les longueurs des côtés opposés et adjacents d'un triangle rectangle et c la longueur de l'hypoténuse (le côté le plus long), alors $a^2 + b^2 = c^2$. Écrivez un programme qui demande à l'utilisateur de rentrer ces trois nombres et qui détermine s'ils peuvent ou non représenter les côtés d'un triangle rectangle.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

7

RÉPÉTITIONS : UNE ÉTUDE APPROFONDIE DES BOUCLES

Vous avez déjà utilisé certaines structures de répétition de Scratch, mais ce chapitre va les étudier plus en détail. Vous y découvrirez de nouveaux blocs permettant de créer des boucles simples, des boucles imbriquées et des programmes récursifs. Vous aborderez notamment les concepts suivants :

- les structures de répétition qui permettent d'exécuter des instructions de manière répétée ;
- la validation des entrées utilisateur ;
- les boucles comptées et conditionnelles ;
- les procédures récursives, qui peuvent s'appeler elles-mêmes.

Bien que nous trouvions les tâches récursives ennuyeuses, les ordinateurs semblent les adorer. Les *structures de répétition*, mieux connues sous le

Copyright © 2016 Eyrolles.

nom de *boucles*, sont des commandes de programmation qui indiquent à l'ordinateur d'exécuter des instructions de manière répétée. Le type le plus simple est la *boucle définie*, qui répète une séquence d'instructions un certain nombre de fois ; elle est également appelée *boucle comptée*. Un autre type, la *boucle conditionnelle* ou *boucle indéfinie*, répète des instructions jusqu'à ce qu'un certain événement survienne. Un troisième type, appelé *boucle infinie*, répète des instructions indéfiniment.

Dans ce chapitre, vous étudierez les différentes structures de répétition disponibles dans Scratch. J'expliciterai les boucles comptées et conditionnelles, puis je vous présenterai le bloc **stop**, qui permet d'arrêter une boucle infinie. Vous verrez également comment utiliser des boucles pour valider les entrées de l'utilisateur.

Ce chapitre traite également des *boucles imbriquées* (des boucles qui contiennent d'autres boucles) et en montre plusieurs exemples d'utilisation. Nous parlerons également de la *récursivité* – une procédure qui s'appelle elle-même – qui est une autre moyen d'effectuer des répétitions. Enfin, nous développerons un certain nombre d'applications intéressantes utilisant boucles et conditions, et nous verrons comment incorporer des boucles dans des programmes pratiques.

Plus de blocs de boucles

Comme vous l'avez découvert au chapitre 2, les blocs de boucle permettent de répéter une commande ou un ensemble de commandes dans un programme. Scratch propose les trois types de blocs de répétition présentés à la figure 7-1.



Figure 7-1 : Les blocs de répétition de Scratch

Vous avez déjà utilisé le bloc **répéter** et le bloc **répéter indéfiniment** lors de nombreux exemples dans ce livre. Dans cette section, nous étudierons le bloc **répéter jusqu'à** et expliquerons une partie des termes techniques associés aux boucles en général.

Chaque répétition d'une boucle est appelée une *itération*. Le bloc **répéter**, que vous connaissez bien, est une boucle comptée car il répète des commandes un certain nombre de fois. Nous utilisons généralement cette boucle lorsque nous connaissons le nombre de répétitions à effectuer, comme lorsque nous voulons construire un polygone avec un nombre connu de côtés.

Le bloc **répéter jusqu'à** est un bloc conditionnel. Les instructions qu'il contient sont répétées jusqu'à ce que sa condition test soit vraie. Ce bloc est utilisé lorsque nous n'avons aucune connaissance du nombre de répétitions à effectuer. Vous pouvez dire, par exemple, « répéter la commande **demander** tant que l'utilisateur entre un nombre positif » ou bien « répéter le tir

Copyright © 2016 Eyrolles.

du missile tant que l'énergie du joueur n'est pas tombée en dessous d'une certaine valeur ». Les sections suivantes fournissent une explication détaillée des boucles conditionnelles.

Le bloc répéter jusqu'à

Imaginons que nous développions un jeu qui présente à l'utilisateur une question de mathématiques de niveau primaire. Si la réponse du joueur est incorrecte, le jeu lui repose la question pour lui donner une autre chance. En d'autres termes, le jeu pose la même question *jusqu'à* ce que le joueur réponde juste. Le bloc **répéter** n'est clairement pas approprié dans ce cas car vous ne savez pas à l'avance combien d'essais il faudra au joueur avant de trouver la bonne réponse ; il pourrait aussi bien réussir au premier essai qu'au centième. Le bloc **répéter jusqu'à** est utile dans de tels scénarios (figure 7-2).

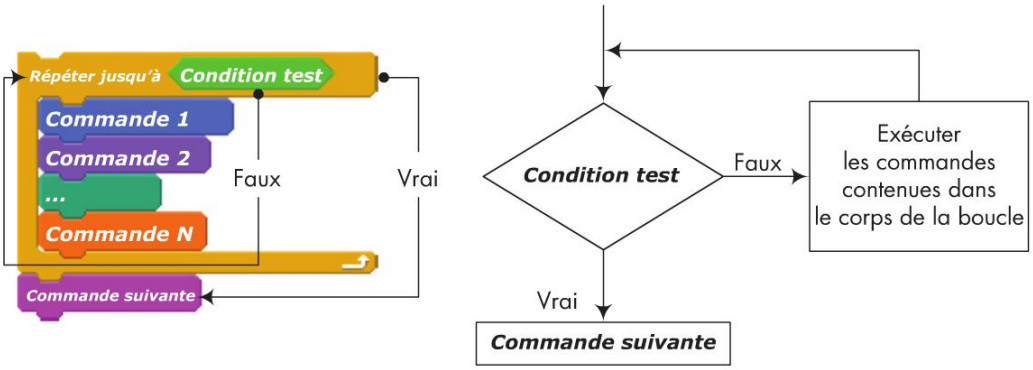


Figure 7-2 : Le bloc **répéter jusqu'à** *exécute une série d'instructions de façon répétée jusqu'à ce qu'une condition soit vraie.*

Ce bloc contient une expression booléenne dont la valeur est testée à l'entrée de la boucle. Si l'expression est fausse, les commandes contenues dans le corps de la boucle sont exécutées. Lorsque la dernière commande est exécutée, la boucle redémarre et l'expression est à nouveau testée. Si elle est toujours fausse, le corps de la boucle est exécuté à nouveau, et ce *jusqu'à* ce que l'expression test soit vraie. Le programme passe alors directement à la commande qui suit la boucle.

Notez que si la condition test est déjà vraie à la première itération, les commandes contenues dans la boucle ne sont pas exécutées. De même, le bloc **répéter jusqu'à** ne s'arrête pas avant qu'une commande (comprise dans la boucle ou dans un autre script actif) ne rende la condition vraie. Si le résultat de la condition test ne devient jamais vrai, nous entrons alors dans une boucle infinie.

La figure 7-3 montre un exemple pratique de l'utilisation de ce bloc. Tant que le lutin Joueur est à plus de 100 pas du lutin Garde, ce dernier continue de se déplacer horizontalement, en rebondissant sur les bords gauche et droit de la scène. Si la distance qui les sépare devient inférieure à 100, le bloc **répéter jusqu'à** se termine et le Garde commence à chasser le Joueur. Le code de la chasse n'est pas présenté sur la figure. Le bloc **distance de** se trouve dans la palette *Capteurs*.

Copyright © 2016 Eyrolles.



Figure 7-3 : Un exemple simple montrant le bloc **répéter jusqu'à** *en action*

Chasse.sb2

FAITES LE TEST 7-1

Ouvrez l'application *Chasse.sb2* et exécutez-la. Utilisez les flèches du clavier pour déplacer le lutin Joueur près du lutin Garde et voir la chasse en action. Comment changeriez-vous la condition test pour libérer le Garde lorsque la position *y* du Joueur sort d'une certaine zone (allant par exemple de -50 à 50) ? Implémentez ce changement et vérifiez votre solution.

Construire un bloc si infini

Les boucles infinies sont utiles dans beaucoup de situations de programmation. Par exemple, dans les chapitres précédents, vous avez utilisé le bloc **répéter indéfiniment** pour jouer une musique de fond et pour animer des lutins en changeant leurs costumes continuellement. Ce bloc est une *boucle infinie inconditionnelle* car aucune condition ne contrôle l'exécution des commandes qu'elle contient.

Vous pouvez cependant changer cela assez facilement en y imbriquant un bloc **si** pour créer une *boucle infinie conditionnelle* (figure 7-4). La condition test du bloc **si** est vérifiée à chaque itération et les commandes qui y sont incluses ne s'exécutent que si elle est vraie. Notez que, puisque le bloc **répéter indéfiniment** est supposé s'exécuter pour toujours, vous ne pouvez pas attacher de blocs en dessous, car ils ne s'exécuteraient jamais.

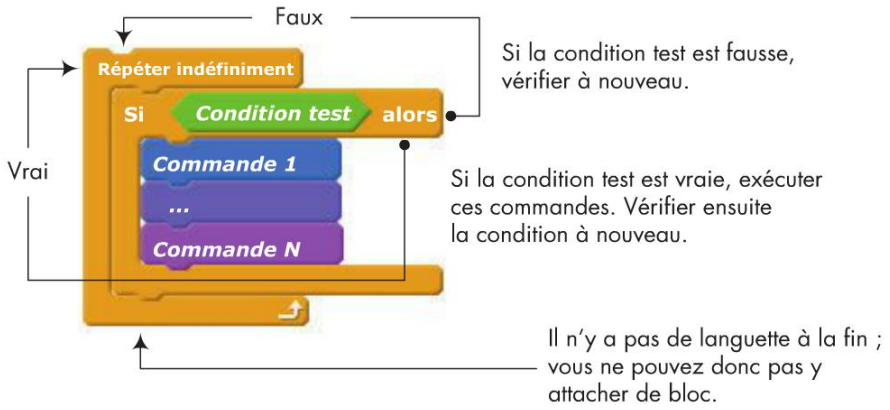


Figure 7-4 : Vous créez une boucle infinie conditionnelle en combinant un bloc **répéter indéfiniment** *avec un bloc* **si**.

Copyright © 2016 Eyrolles.

La structure **répéter indéfiniment/si** combinée est fréquemment utilisée pour contrôler les mouvements du lutin avec les flèches du clavier (figure 7-5).

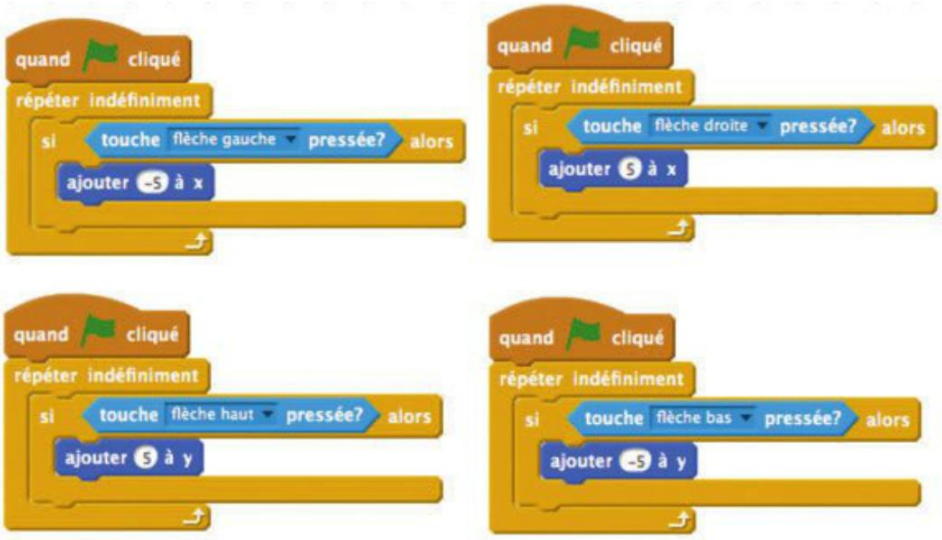


Figure 7-5 : Ces scripts vous permettent de déplacer un lutin en utilisant les flèches du clavier. Chaque script répond à l'une de ces quatre touches.

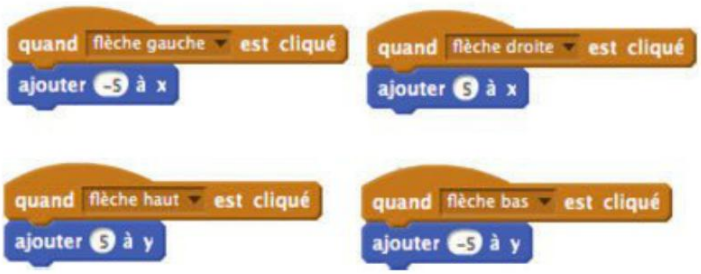
FlèchesClavier1.sb2

Lorsqu'on clique sur le drapeau vert, les quatre flèches directionnelles du clavier sont gérées dans quatre boucles infinies indépendantes. Quand l'une des flèches est pressée, la boucle correspondante effectue un changement dans la position *x* ou *y* du lutin.

Créez ces scripts dans Scratch (ou ouvrez le fichier *FlèchesClavier1.sb2*) et exécutez le programme. Vous remarquerez que, si vous appuyez sur les flèches du haut et de droite simultanément, votre lutin se déplacera en diagonale dans la direction nord-ouest. Essayez d'autres combinaisons de flèches pour voir comment répond l'application.

FAITES LE TEST 7-2

Une autre manière de contrôler les mouvements du lutin avec les flèches est représentée ci-après. Comparez cette méthode avec celle présentée à la figure 7-5. Laquelle des deux est la plus réactive à l'appui sur une touche ? Comment se comporte le script ci-après lors de l'appui simultané sur plusieurs flèches (haut et droite par exemple) ? Essayez à présent de placer les quatre blocs **si** de la figure 5-7 dans une seule boucle **répéter indéfiniment** et appuyez sur deux flèches simultanément. Comment cela change-t-il le comportement du lutin ?



Copyright © 2016 Eyrolles.

Les commandes d'arrêt (stop)

Imaginons que vous écriviez un programme qui trouve le premier entier inférieur à 1 000 divisible par 3, 5 et 7. Vous pourriez écrire un script qui teste dans une boucle tous les nombres de 0 à 1 000. Dans ce cas, vous voudrez *arrêter* la recherche lorsque vous trouverez le nombre que vous cherchez (105 dans cet exemple).

Comment dire à Scratch d'arrêter la boucle et les scripts ? C'est la commande **stop** (de la palette *Contrôle*) qui arrête les scripts actifs. Le menu déroulant fournit trois options montrées sur la figure 7-6.



Figure 7-6 : Utilisation de la commande **stop** *dans Scratch*

La première option met immédiatement fin au script qui l'a appelée. La seconde option, quant à elle, met fin à tous les scripts actifs de l'application ; elle est équivalente au bouton stop rouge à côté du drapeau vert en haut de la scène. Notez que vous ne pouvez pas attacher de commande sous le bouton stop lorsqu'il est utilisé pour l'une de ces deux options.

StopDemo.sb2

La troisième option du bloc **stop** permet au lutin ou à la scène de mettre fin à tous ses scripts excepté celui qui a appelé la commande **stop**. Avec cette option, le bloc prend la forme d'un bloc de pile et vous pouvez donc y attacher des instructions qui seront exécutées ensuite. Voyons cette commande en action dans un jeu simple, illustré sur la figure 7-7.

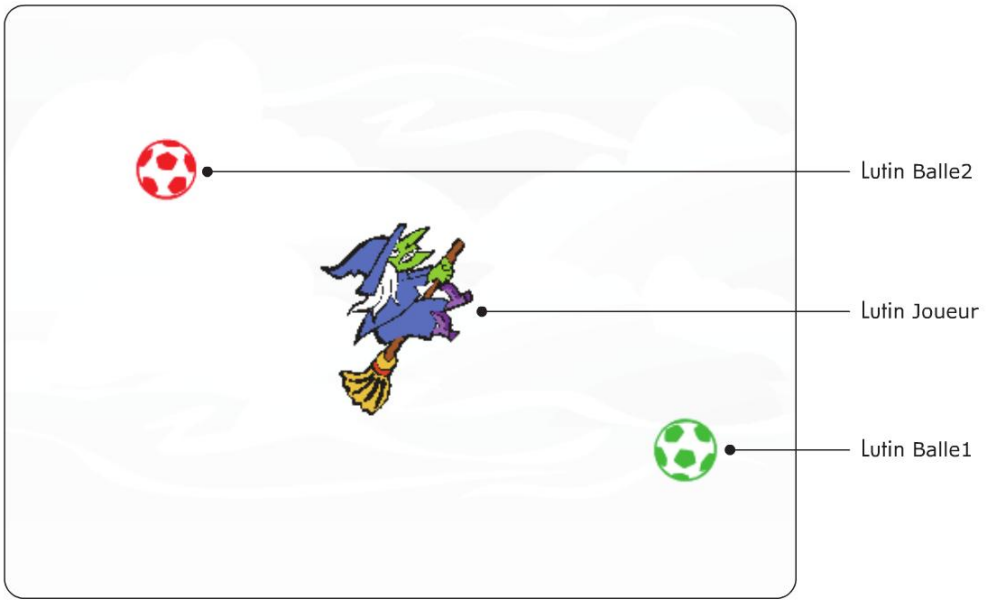


Figure 7-7 : Dans ce jeu, le joueur déplace la sorcière sur la scène en essayant d'éviter les balles.

Copyright © 2016 Eyrolles.

Les deux balles de la figure se déplacent sur la scène et chassent la sorcière. Le joueur déplace la sorcière avec les flèches du clavier et essaie d'éviter de se faire toucher par l'une des deux balles. Si la balle rouge touche le joueur, le jeu s'arrête. Si c'est la balle verte, elle cesse de chasser le joueur, mais la balle rouge se déplace légèrement plus vite – ce qui rend l'évasion plus difficile. Les scripts pour déplacer la sorcière sont similaires à ceux de la figure 7-5, je ne les présenterai donc pas ici. Les scripts qui déplacent chaque balle sont présentés sur la figure 7-8.

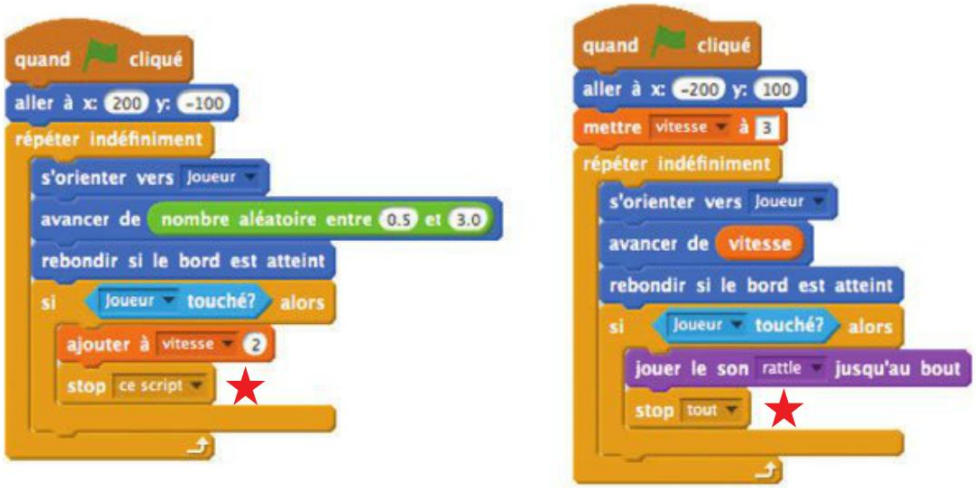


Figure 7-8 : Scripts pour la balle verte (à gauche) et la balle rouge (à droite)

Lorsque la balle verte touche le joueur, elle augmente la variable vitesse (qui définit la vitesse des déplacements de la balle rouge), puis appelle le bloc **stop ce script** pour mettre fin à son script. Tous les autres scripts du jeu devraient continuer à s'exécuter normalement. L'utilisation de cette commande fonctionne bien ici car nous ne voulons augmenter la vitesse de la balle rouge qu'une seule fois. Si la balle rouge touche le joueur, cependant, elle exécute la commande **stop tout**, qui cause l'arrêt de tout script actif de l'application.

FAITES LE TEST 7-3

Ouvrez ce jeu et voyez comment il fonctionne. Regardez ce qui arrive aux bords jaunes des deux scripts de la figure 7-8 lorsque les balles verte et rouge touchent la sorcière.

Vous pouvez aussi utiliser le bloc **stop** pour mette fin à une procédure et faire retourner l'exécution à l'instruction suivante dans le script appelant. La section suivante montre ce concept en action.

Copyright © 2016 Eyrolles.

Mettre fin à une boucle

Recherche DeNombre .sb2

Imaginons que vous vouliez trouver la première puissance de 2 supérieure à 1 000. Nous écrirons une procédure qui évaluera dans une boucle les nombre 2^1, 2^2, 2^3, 2^4 et ainsi de suite. Lorsque nous trouverons le nombre dont nous avons besoin, nous voudrons que le programme nous donne la réponse et mette fin à la procédure. La figure 7-9 montre deux manières d'implémenter cette approche.

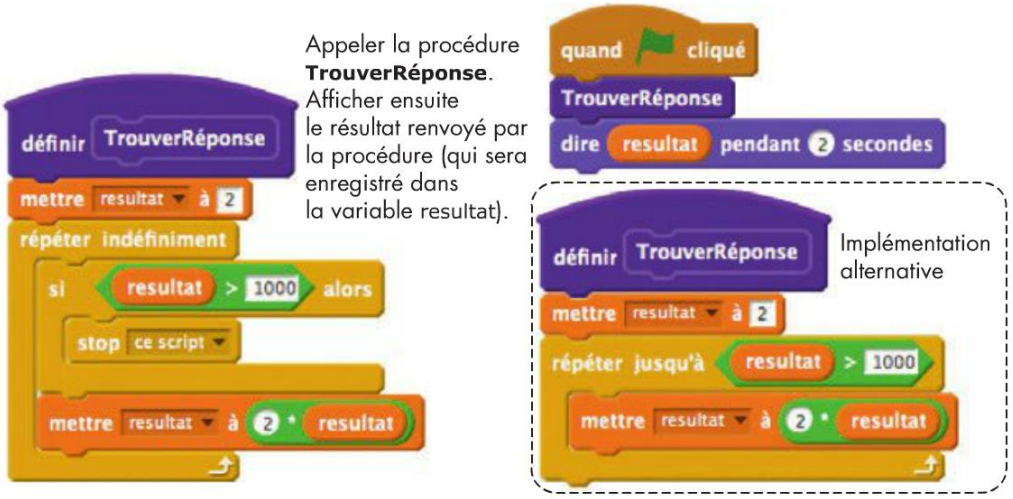


Figure 7-9 : Deux manières de trouver la première puissance de 2 supérieure à 1 000

La procédure à gauche de la figure 7-9 initialise la variable resultat à 2, qui est la première puissance de 2 à évaluer, et entre dans une boucle infinie, à la recherche de la réponse. Elle vérifie la valeur de la variable resultat à chaque itération de la boucle. Si elle est supérieure à 1 000, la procédure appelle la commande **stop ce script** pour s'arrêter et pour retourner dans le script appelant. Sinon, la commande qui suit le bloc **si** (qui multiplie la valeur précédente de resultat par 2) est exécutée et la prochaine itération de la boucle commence. Si vous suivez le cheminement de cette procédure, vous verrez que le bloc **si** trouve resultat égal à 2 lors de la première itération, 4 dans la seconde, 8 dans la troisième et ainsi de suite. Cela continue jusqu'à ce que resultat soit supérieur à 1 000 ; à ce stade, la procédure s'arrête et l'exécution retourne au script appelant, qui affiche le résultat en utilisant le bloc **dire**.

La figure 7-9 (à droite) montre une autre manière d'implémenter cette procédure. Nous utilisons ici un bloc **répéter jusqu'à** qui continue l'exécution de la boucle jusqu'à ce que resultat, multiplié par 2 à chaque itération, soit supérieur à 1 000. Quand cela arrive, la boucle prend fin naturellement et l'exécution retourne au script ayant appelé la procédure. Notez que nous n'avons pas besoin d'utiliser le bloc **stop** dans ce cas.

Le bloc **stop** peut également servir pour valider des entrées utilisateur, comme vous le verrez dans la section suivante.

Valider les entrées utilisateur

Quand vous écrirez des applications qui lisent des données entrées par l'utilisateur, vous devrez vérifier que ces dernières sont valides avant de

Copyright © 2016 Eyrolles.

commencer à les traiter. Les structures de répétition peuvent vous aider à accomplir cette tâche. Si l'entrée est invalide, vous utilisez une boucle pour afficher un message d'erreur approprié et demander à l'utilisateur de rentrer une nouvelle valeur.

Imaginons que vous développiez un jeu à deux niveaux et que vous vouliez demander à l'utilisateur celui dans lequel il veut jouer. Les seules entrées valides dans ce cas sont 1 et 2. Si l'utilisateur rentre quoi que ce soit d'autre, vous voudrez lui offrir une autre chance de choisir son niveau (figure 7-10).

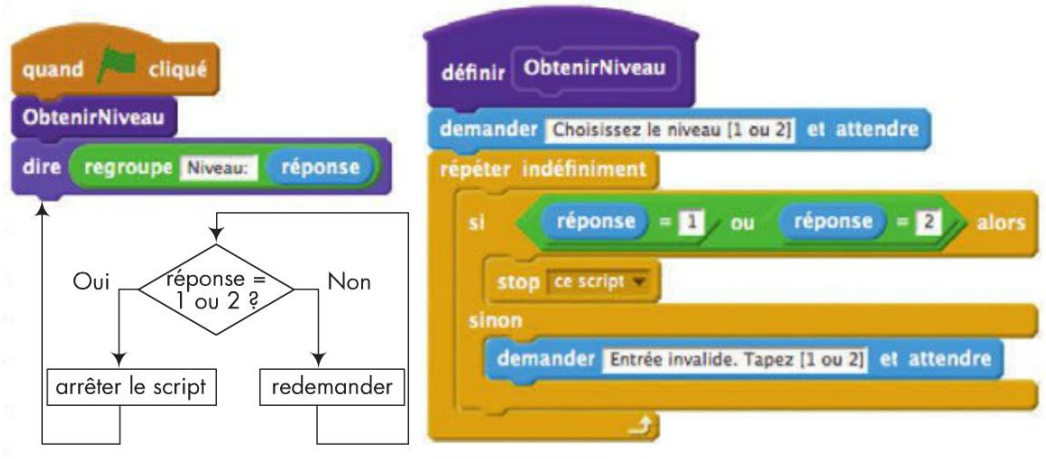


Figure 7-10 : Validation des entrées en utilisant le bloc **répéter indéfiniment**

La procédure **ObtenirNiveau** demande à l'utilisateur d'entrer un choix et vérifie la réponse dans une boucle **répéter indéfiniment**. Si l'utilisateur entre un nombre invalide, la boucle lui redemande de rentrer un nombre. S'il entre un nombre valide, la commande **stop ce script** sort de la boucle en mettant fin à la procédure. À ce moment-là, le script principal, qui attendait patiemment que la procédure **ObtenirNiveau** se termine, reprend l'exécution à la commande **dire**. La figure 7-11 montre comment faire la même chose avec un bloc **répéter jusqu'à**.

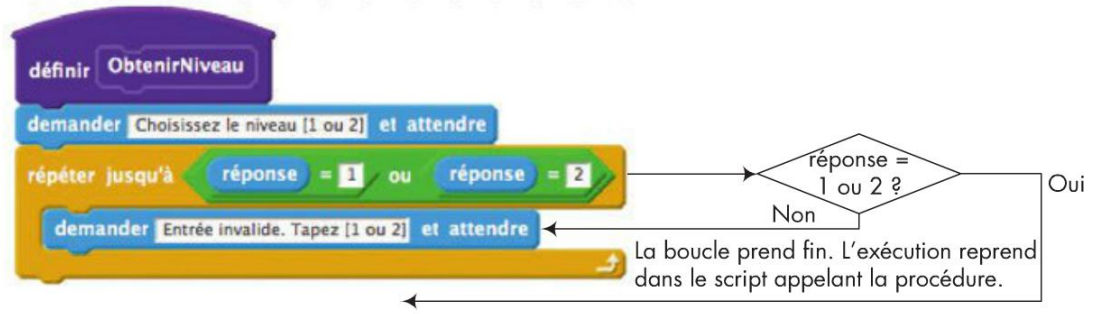


Figure 7-11 : Validation de l'entrée utilisateur avec le bloc **répéter jusqu'à**

La procédure de la figure 7-11 demande à l'utilisateur d'entrer un choix et attend sa réponse. Si c'est 1 ou 2, la condition dans l'en-tête du bloc **répéter jusqu'à** est vraie, ce qui termine la boucle naturellement et mène à la fin de la procédure. En revanche, si l'utilisateur entre quoi que ce soit d'autre que 1 ou 2, la condition de la boucle est fausse et la commande **demander** de l'intérieur de la boucle est exécutée. Cette commande attend l'entrée

Copyright © 2016 Eyrolles.

utilisateur à nouveau et le bloc **répéter jusqu'à** continue de demander une réponse jusqu'à ce que l'utilisateur entre un choix valide. À nouveau, notez que cette implémentation n'a pas besoin du bloc **stop**.

Les compteurs

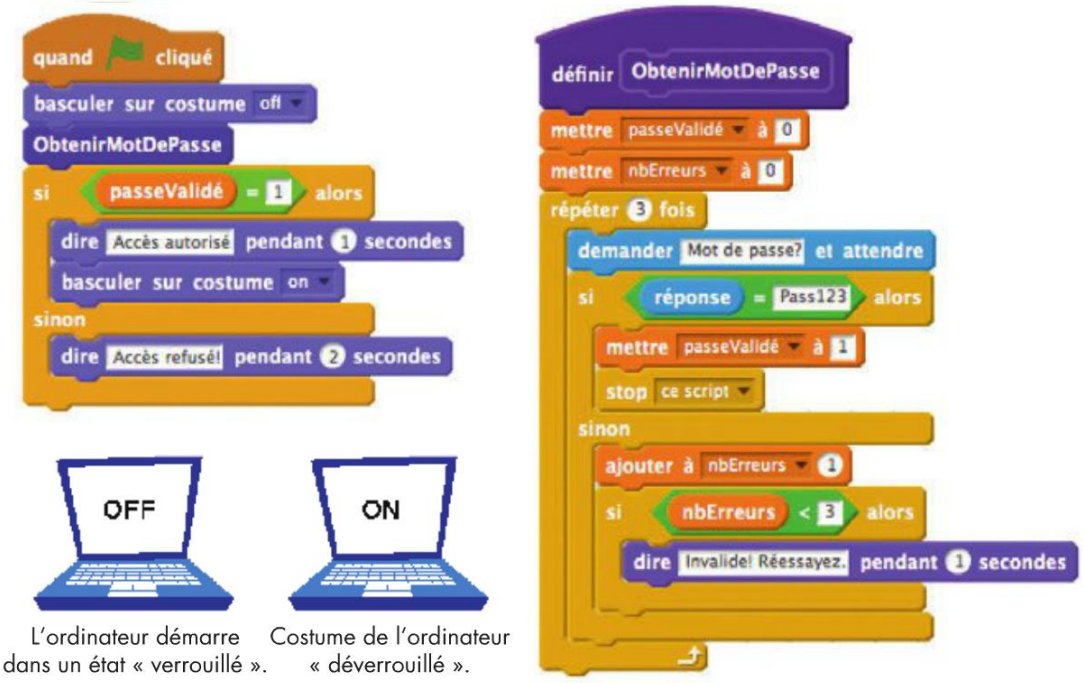
Vous aurez parfois besoin de garder une trace du nombre d'itérations qu'une boucle effectue. Par exemple, si vous voulez ne donner que trois chances aux utilisateurs pour entrer le bon mot de passe, vous devrez compter le nombre d'essais et les bloquer au bout du troisième.

Vous pouvez gérer un tel scénario en utilisant une variable (généralement appelée compteur) qui mémorise le nombre d'itérations. Étudions quelques exemples qui mettront en pratique l'utilisation des compteurs.

Vérifier un mot de passe

Vérification DeMotDePasse .sb2

Le programme de la figure 7-12 demande à l'utilisateur d'entrer un mot de passe pour déverrouiller l'ordinateur. Le lutin Ordinateur possède deux costumes : l'image off indique que l'ordinateur est verrouillé, tandis que l'image on indique qu'il est déverrouillé. L'utilisateur peut se voir refuser l'accès s'il rentre trois mots de passe invalides.



L'ordinateur démarre dans un état « verrouillé ».

Costume de l'ordinateur « déverrouillé ».

Figure 7-12 : Le script donne à l'utilisateur trois chances de rentrer le bon mot de passe.

Lorsqu'on clique sur le drapeau vert, le lutin Ordinateur met le costume off et appelle la procédure **ObtenirMotDePasse** pour authentifier l'utilisateur. Cette procédure est censée renvoyer le résultat de la vérification du mot de passe au script principal en utilisant le drapeau passeValidé. Le bloc **si/sinon** vérifie le drapeau pour décider si l'utilisateur a l'autorisation d'accéder à l'ordinateur ou non. Si passeValidé est à 1, le mot de passe entré par

Copyright © 2016 Eyrolles.

l'utilisateur est correct et le bloc **si** exécute la commande **dire** pour afficher *Accès autorisé* puis met à l'ordinateur le costume on. Sinon, le script affiche *Accès refusé !* et le lutin garde son costume initial off.

La procédure **ObtenirMotDePasse** règle le drapeau passeValidé à 0 pour indiquer qu'elle n'a pas encore reçu de mot de passe valide et initialise notre compteur, la variable nbErreurs, à 0. Elle exécute ensuite une boucle **répéter** avec un nombre maximal de répétitions de 3. À chaque itération, la boucle demande à l'utilisateur d'entrer un mot de passe. Si c'est le bon (Pass123 dans cet exemple), le drapeau passeValidé est mis à 1, la procédure s'arrête d'elle-même en appelant la commande **stop ce script** et l'exécution retourne au script appelant. En revanche, si l'utilisateur n'a pas utilisé ses trois chances et rentre un mot de passe erroné, un message d'erreur s'affiche et l'invite à réessayer. Si l'utilisateur se trompe à trois reprises consécutives, la boucle **répéter** se termine automatiquement et la procédure renvoie au script appelant la valeur du drapeau passeValidé, qui sera restée à 0.

FAITES LE TEST 7-4

Ouvrez cette application et exécutez-la. Que se passe-t-il si vous entrez le mot de passe paSS123 au lieu de Pass123 ? Quelle information cela vous donne-t-il sur les comparaisons dans Scratch ? Essayez d'implémenter la procédure **ObtenirMotDePasse** en utilisant un bloc **répéter jusqu'à**.

Compter avec un incrément constant

CompterAvecMontantConst.sb2

Bien sûr, vous n'êtes pas toujours obligé d'incrémenter votre compteur de 1 dans une boucle. Le script ❶ de la figure 7-13 possède un compteur évoluant de 5 à 55 en s'incrémentant de 5. Le script ❷ fait décompter le compteur de 99 à 0 en décrémentant de 11 à chaque itération.



Figure 7-13 : Vous pouvez incrémenter ou décrémenter des compteurs avec une valeur différente de 1.

Pour voir cette technique de comptage dans une application pratique, la figure 7-14 montre un script qui trouve la somme de tous les entiers pairs allant de 2 à 20 (donc 2 + 4 + 6 + 8 + … + 20).

Copyright © 2016 Eyrolles.

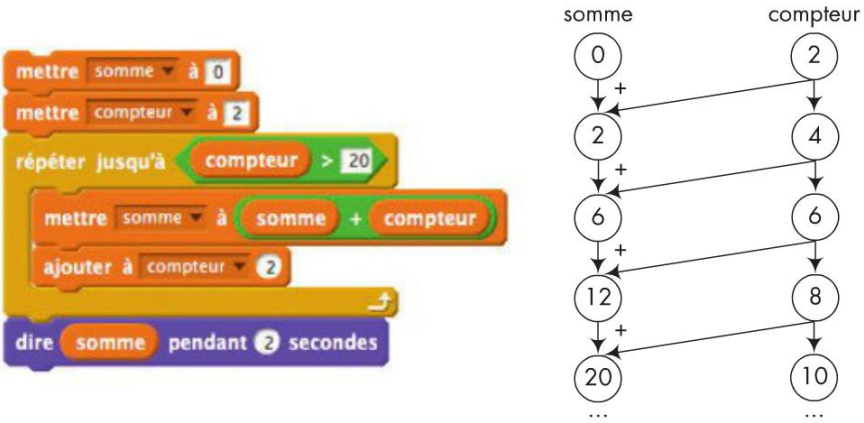


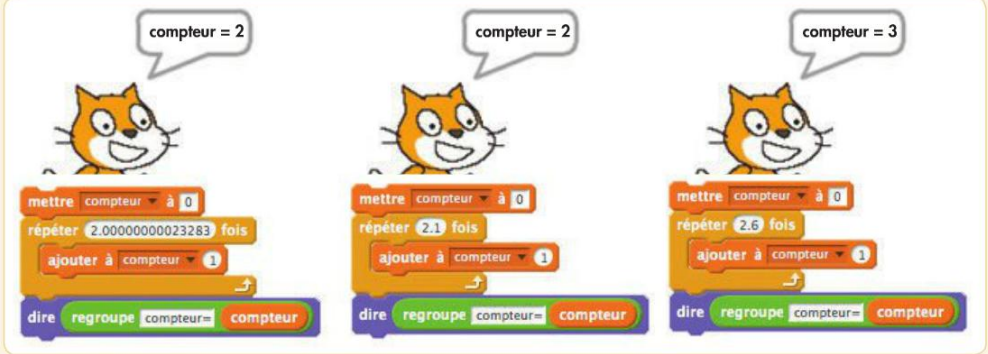
Figure 7-14 : Ce script trouve la somme de tous les entiers pairs allant de 2 à 20.

Ce script commence par initialiser la variable somme à 0 et la variable compteur à 2, puis entre dans une boucle conditionnelle qui se répète jusqu'à ce que compteur dépasse 20. À chaque itération, la variable compteur est additionnée à la somme actuelle puis incrémentée de 2 pour obtenir le prochain nombre pair. Essayez de deviner quelle sera la sortie de ce script et exécutez-le pour vérifier votre réponse.

Répéter NonEntier .sb2

RÉPÉTER UN NOMBRE NON ENTIER DE FOIS

À votre avis, que se passera-t-il si vous demandez à Scratch de répéter une boucle 2,5 fois ? Les trois exemples présentés ci-après montrent comment Scratch gère les répétions d'un nombre non entier de fois.



Bien sûr, on ne peut pas répéter quelque chose 2,5 fois, mais Scratch ne vous empêchera pas de le faire. Plutôt que de vous donner un message d'erreur, Scratch arrondira ce nombre automatiquement à l'entier près.

Copyright © 2016 Eyrolles.

Retour sur les boucles imbriquées

Rappelez-vous les « carrés rotatifs » que nous avions dessinés en utilisant des boucles imbriquées (chapitre 2). Une boucle (la *boucle interne*) se chargeait de dessiner le carré, tandis que l'autre (la *boucle externe*) contrôlait le nombre de rotations. Dans cette section, vous apprendrez à utiliser le concept de compteurs conjoints avec des boucles imbriquées pour créer des itérations dans deux (ou plus) dimensions. Cette technique est une partie essentielle de la programmation et sert à résoudre un large panel de problèmes de programmation.

Pour démarrer, imaginons qu'un restaurant local propose quatre types de pizzas (P1, P2, P3 et P4) et trois types de salades (S1, S2 et S3). Si vous mangiez à cet endroit, vous auriez donc à choisir entre 12 combinaisons possibles : pizza P1 avec l'une des trois salades, P2 avec l'une des trois salades et ainsi de suite. Le propriétaire du restaurant voudrait imprimer un menu qui fasse la liste des combinaisons disponibles accompagnées de leur prix et de leur contenu calorique. Voyons comment les boucles imbriquées sont utilisables pour générer cette liste (je vous laisserai comme exercice le calcul des prix et des contenus caloriques).

En y réfléchissant, vous remarquerez que deux boucles imbriquées suffiront : la boucle externe pour les types de pizza et la boucle interne pour les types de salade. La boucle externe commence par P1, puis la boucle interne passe par S1, S2 et S3. Ensuite, la boucle externe passe à P2 et la boucle interne repasse par S1, S2 et S3 et ainsi de suite jusqu'à être passé par tous les types de pizzas (figure 7-15).

Boucles Imbriquées1 .sb2



Figure 7-15 : Visualisation des boucles imbriquées. La variable P *contrôle la boucle externe et la variable* S *contrôle la boucle interne.*

Ce script utilise deux boucles et deux compteurs. Le compteur de la boucle externe s'appelle P et celui de la boucle interne s'appelle S. Dans la première itération de la boucle externe (quand P=1), la valeur du compteur S est mise à 1 et la boucle interne se répète trois fois. À chaque fois, elle exécute la commande **dire** pour afficher les valeurs actuelles de P et S, puis incrémente S de 1. La première itération de boucle externe fait donc dire au lutin « P1,S1 » puis « P1,S2 » puis « P1,S3 ».

Après avoir bouclé trois fois, la boucle interne se termine, P est incrémenté de 1 et la seconde itération de la boucle externe démarre. Le lutin dit

Copyright © 2016 Eyrolles.

donc « P2,S1 » puis « P2,S2 » puis « P2,S3 ». Le processus continue ensuite de la même manière, affichant « P3,S1 » puis « P3,S2 » puis « P3,S3 » et finalement « P4,S1 » puis « P4,S2 » puis « P4,S3 ». Observez bien le script et suivez son cheminement pour vous assurer de bien comprendre comment il fonctionne.

Maintenant que vous savez ce que peuvent faire les boucles imbriquées, utilisons cette technique pour résoudre un problème de mathématiques intéressant. Nous voulons écrire un programme qui trouve trois entiers positifs, n_1, n_2 et n_3 tels que $n_1 + n_2 + n_3 = 25$ et $(n_1)^2 + (n_2)^2 + (n_3)^2 = 243$. Puisque les ordinateurs sont doués pour effectuer des tâches répétitives, notre plan est de tester toutes les combinaisons de nombres possibles (une technique appelée *recherche exhaustive*) et de laisser travailler la machine.

Selon notre première équation, le premier nombre, n_1, peut prendre n'importe quelle valeur comprise entre 1 et 23 puisque nous devrons y ajouter deux nombres pour obtenir 25. Vous avez peut-être déjà remarqué que n_1 ne peut pas dépasser 15 puisque $16^2 = 256$, ce qui est supérieur à 243. Nous ignorerons cependant notre seconde équation pour le moment et définirons tout de même la limite supérieure de la boucle à 23.

Le deuxième nombre, n_2, peut prendre n'importe quelle valeur comprise entre 1 et $24 - n_1$. Par exemple, si n_1 vaut 10, la valeur maximale de n_2 sera 14 puisque n_3 vaut au moins 1. Si nous connaissons n_1 et n_2, nous trouverons $n_3 = 25 - (n_1 + n_2)$. Ensuite, nous devrons vérifier si la somme des carrés de ces trois nombres est égale à 243. Si c'est le cas, nous aurons terminé. Sinon, nous devrons essayer une autre combinaison de n_1 et n_2. Vous pouvez voir le script terminé sur la figure 7-16.

Boucles Imbriquées2 .sb2

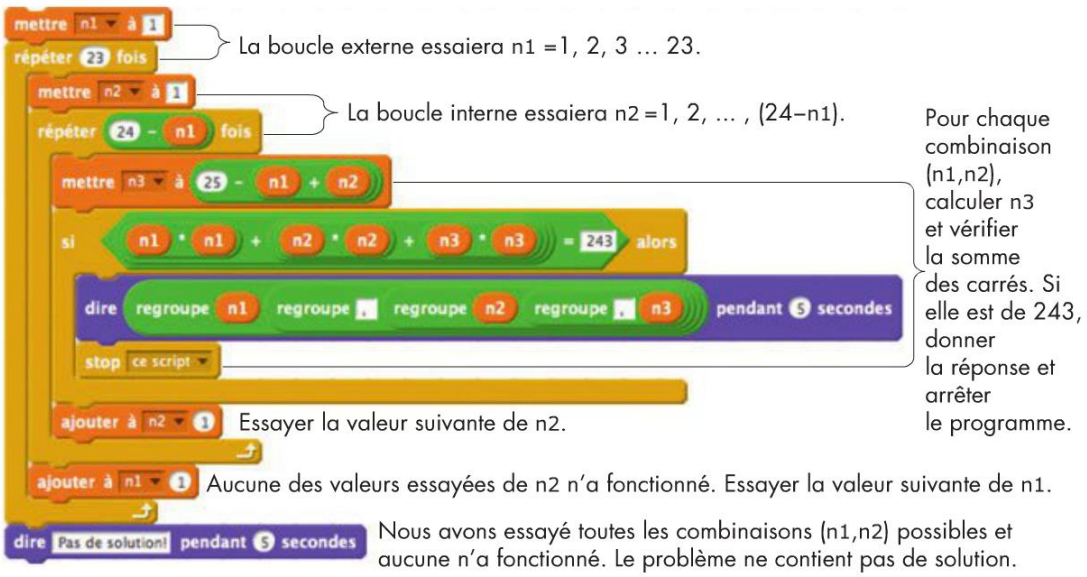


Figure 7-16 : Ce script cherche trois nombres positifs dont la somme donne 25 et la somme des carrés donne 243.

La boucle externe teste toutes les valeurs de n1 de 1 à 23. Pour chaque valeur de n1, la boucle interne teste toutes les valeurs de n2 de 1 à (24 – n1).

Copyright © 2016 Eyrolles.

Pour chaque combinaison de n1 et n2, le script règle n3 à 25 – (n1 + n2) et vérifie si la somme des carrés de ces trois nombres donne 243. Si c'est le cas, il donne la réponse et s'arrête.

FAITES LE TEST 7-5

Créez le script de la figure 7-16 et exécutez-le pour trouver n1, n2 et n3. Si vous étudiez ce script attentivement, vous verrez que certaines combinaisons (n1, n2) sont testées plus d'une fois. Par exemple, les nombres (1,2) sont testés lors de la première itération de la boucle externe, alors que les nombres (2,1) sont testés lors de la deuxième itération. Ces deux tests sont redondants ; un seul est réellement nécessaire. Vous pouvez régler ce problème en faisant commencer la boucle interna à n1 au lieu de 1. Appliquez ces changements au script et exécutez-le pour vous assurer qu'il fonctionne convenablement.

Récursivité : des procédures qui s'appellent elles-mêmes

Récursivité.sb2

Les structures de répétition présentées jusqu'à présent répètent un ensemble de commandes à chaque itération. La *récursivité* est une autre technique puissante de répétition. Elle permet à une procédure de s'appeler directement, ou indirectement à travers une autre procédure (par exemple, A appelle B, B appelle C, puis C appelle A). Il n'est peut-être pas très évident de comprendre pourquoi vous voudriez faire une telle chose mais, en réalité, la récursivité simplifie la solution de beaucoup de problèmes informatiques. Démontrons ce concept en considérant l'exemple simple de la figure 7-17.

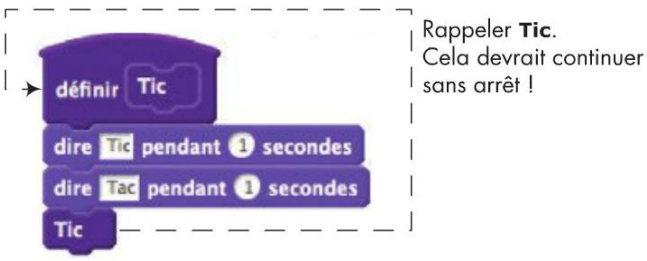


Figure 7-17 : Une procédure récursive

La procédure **Tic** exécute deux commandes **dire** (la première dit « Tic » et la seconde dit « Tac »), puis se rappelle elle-même. Le deuxième appel fait exactement la même chose et le lutin continuera à dire « Tic Tac » sans arrêt si aucune action extérieure ne l'arrête. Le seul moyen de l'arrêter dans ce cas est de cliquer sur le bouton rouge stop, ou d'utiliser le bloc **stop** de la palette *Contrôle*. Utiliser une procédure qui s'appelle elle-même nous a permis de répéter les commandes **dire** sans blocs de boucle. La forme de récursivité utilisée dans cet exemple est appelée *récursivité terminale* car l'appel récursif se trouve à la fin de la procédure. Scratch permet également les appels récursifs avant la dernière ligne, mais nous ne les étudierons pas dans ce livre.

Copyright © 2016 Eyrolles.

Puisque la récursivité infinie n'est en général pas une bonne idée, vous devrez en contrôler l'exécution avec des conditions. Par exemple, la procédure pourrait contenir un bloc **si** qui détermine si l'appel récursif doit être effectif ou non. Pour illustrer cette technique, la figure 7-18 montre une procédure récursive qui décompte à partir d'un nombre initial (spécifié par le paramètre décompte) jusqu'à 0.

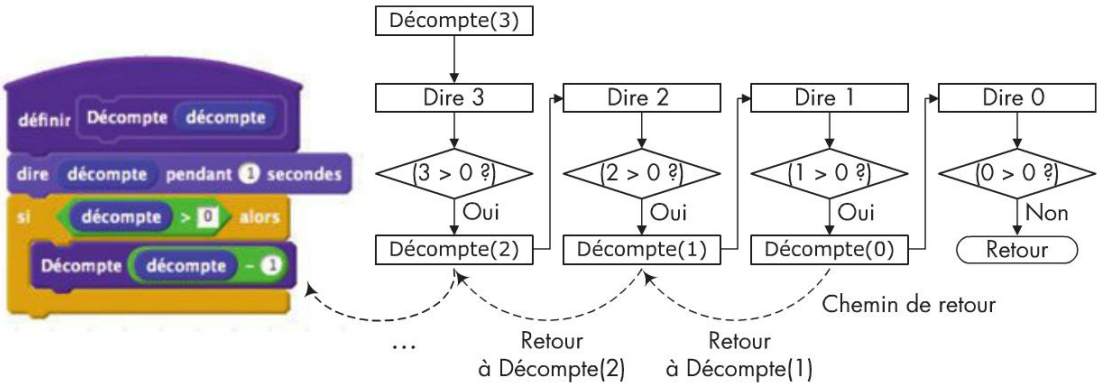


Figure 7-18 : Le bloc **si** *est utilisé pour déterminer si l'appel récursif doit être exécuté ou non.*

Voyons comment **Décompte** fonctionne lorsqu'elle est appelée avec un argument de 3. La procédure démarre en affichant le nombre 3 grâce à la commande **dire**, puis vérifie si décompte est supérieur à 0. Puisque 3 est supérieur à 0, la procédure soustrait 1 à décompte et s'appelle elle-même avec un argument de 2.

Lors du deuxième appel, la procédure affiche 2 et se rappelle avec un argument de 1 puisque 2 est supérieur à 0. Cela continue jusqu'à ce que l'appel de Décompte(0) soit effectué. Après avoir affiché le nombre 0 dans une bulle, la procédure vérifie si décompte est supérieur à 0. Puisque l'expression de l'en-tête du bloc **si** est fausse, l'appel récursif n'est pas exécuté et la procédure se termine. Essayez de suivre le chemin de retour de la procédure présenté à la figure 7-18.

Maintenant que nous avons fait le tour de la récursivité terminale, nous pouvons utiliser ces connaissances dans des applications intéressantes. Considérons, par exemple, la procédure **Lame** présentée sur la figure 7-19.

RécursivitéLame.sb2

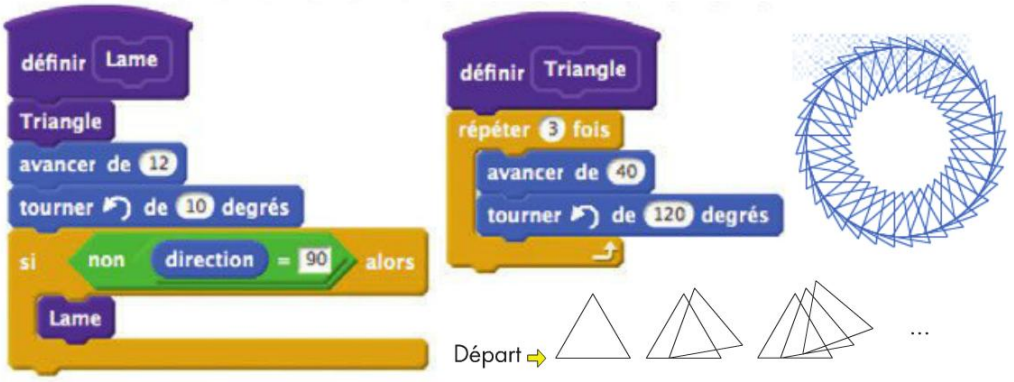


Figure 7-19 : Utilisation de la direction d'un lutin pour arrêter la récursivité

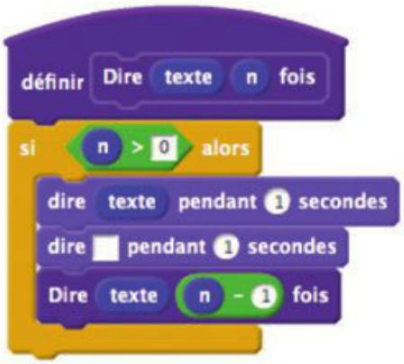
Copyright © 2016 Eyrolles.

Supposons que le lutin qui exécute cette procédure démarre sur la scène avec une direction de 90°. Après avoir dessiné un triangle équilatéral, il avance de 12 pas et tourne de 10° dans le sens inverse des aiguilles d'une montre. La procédure vérifie ensuite la nouvelle direction du lutin. S'il n'est pas orienté à 90°, elle lance un appel récursif d'elle-même pour dessiner le prochain triangle. S'il l'est, l'appel récursif n'a pas lieu et la procédure se termine après avoir dessiné la lame de scie circulaire présentée à la figure 7-19.

Pour les exemples simples présentés ici, il est probablement plus facile d'utiliser un bloc **répéter**. Néanmoins, comme je l'ai dit au début de cette section, la récursivité permet, par rapport à l'itération, de simplifier la résolution de nombreux problèmes de programmation.

FAITES LE TEST 7-6

Que fait la procédure suivante ? Implémentez-la et appelez-la avec différents arguments pour vérifier votre réponse.



Projets Scratch

Maintenant que vous savez vous servir des répétitions dans un script Scratch, il est temps de mettre en pratique ce que vous avez appris dans ce chapitre. Dans cette section, je vous guiderai à travers plusieurs projets qui amélioreront votre compréhension de la programmation et vous donneront de l'inspiration pour vos futurs projets.

Horloge analogique

Horloge Analogique .sb2

Le bloc **actuel** de la palette *Capteurs* fournit l'année, le mois, le jour de la semaine, l'heure, la minute ou la seconde actuelle, selon votre sélection dans le menu déroulant. Notre premier projet utilise ce bloc pour implémenter l'horloge analogique présentée à la figure 7-20. L'application contient quatre lutins : Sec, Min et Heure représentent les trois aiguilles et Horaire (un petit point blanc) affiche l'heure dans un format numérique (dans la bulle de la figure).

Copyright © 2016 Eyrolles.

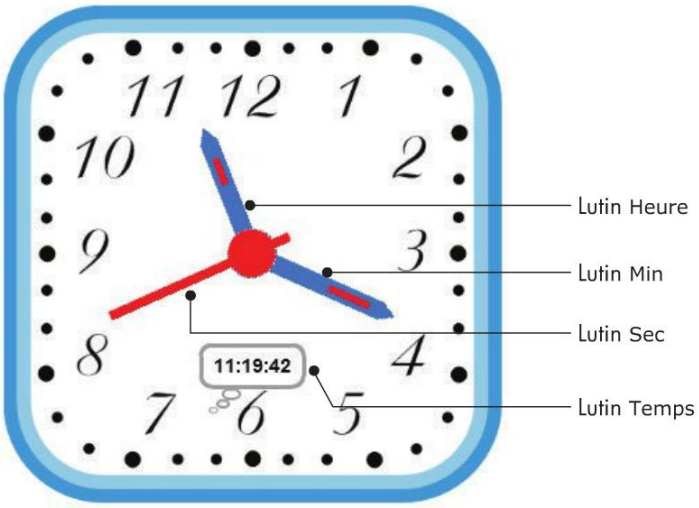


Figure 7-20 : L'application Horloge analogique

L'horloge démarre son exécution lorsqu'on clique sur le drapeau vert. Les quatre lutins rentrent alors dans une boucle **répéter indéfiniment** pour mettre à jour leur position selon l'heure actuelle du système. Les scripts des lutins Sec et Min sont représentés à la figure 7-21

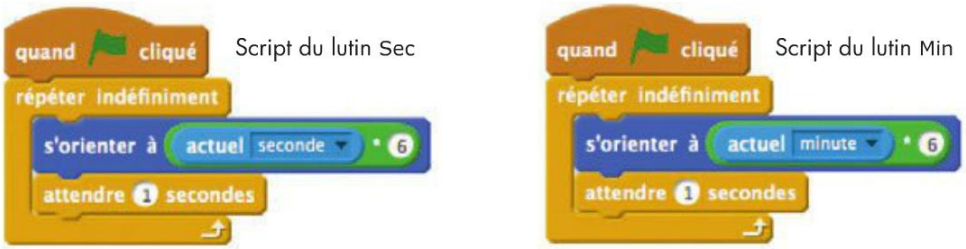


Figure 7-21 : Les scripts des lutins Sec *et* Min

Le nombre de secondes ou de minutes obtenu avec le bloc **actuel** est compris entre 0 et 59. Lorsque le système indique 0 s, le lutin Sec doit être orienté vers le haut (vers 0°). À 15 s, il doit pointer vers la droite (90°) et ainsi de suite. À chaque seconde, l'aiguille Sec doit tourner de 6° (360° divisés par 60 s) dans le sens des aiguilles d'une montre. Un raisonnement similaire est appliqué à l'aiguille Min. Si vous regardez fonctionner l'horloge, vous voyez l'aiguille Sec sauter à chaque seconde et l'aiguille Min à chaque minute. Regardons à présent le script du lutin Heure, présenté à la figure 7-22.

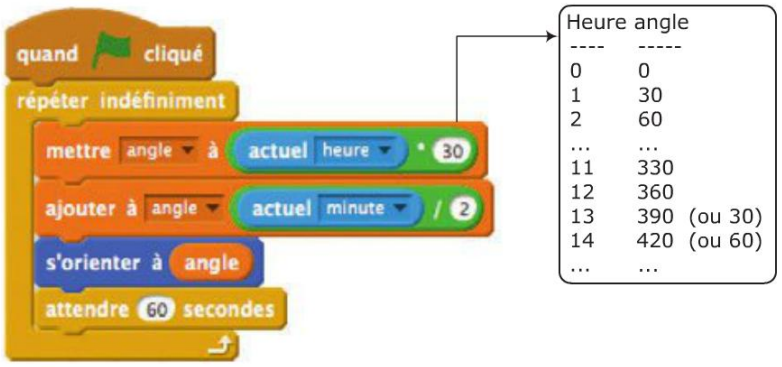


Figure 7-22 : Le script du lutin Heure

Copyright © 2016 Eyrolles.

Le bloc **actuel(heure)** renvoie un nombre compris entre 0 et 23 représentant l'heure de l'horloge du système. Nous voulons que le lutin Heure s'oriente à 0° (vers le haut) pour l'heure 0, 30° pour l'heure 1, 60° pour l'heure 2 et ainsi de suite, comme le montre la figure. Cependant, s'il est par exemple 11 h 50, nous ne voulons pas que l'aiguille des heures pointe sur le 11, mais quelque part proche du 12. Nous réalisons cet ajustement en prenant en compte la valeur des minutes.

Puisqu'une heure (ou 60 min) correspond à 30° sur une horloge, chaque minute vaut 2°. Ainsi, à chaque minute, nous devrons ajuster l'angle du lutin Heure par le nombre actuel de minutes divisé par 2, comme le montre le script.

Le script du lutin Temps est trivial et n'est pas présenté ici. Il utilise des blocs **regroupe** imbriqués pour construire une chaîne de caractères de la forme *heure*:*min*:*sec* et l'afficher dans une bulle (figure 7-20).

FAITES LE TEST 7-7

Ouvrez l'application et exécutez-la. Modifiez le script du lutin Min pour le faire bouger de façon continue, au lieu de sauter à chaque minute. Indice : utilisez la même idée que nous avons appliquée au lutin Heure pour rendre les mouvements de son aiguille continus. Ensuite, modifiez le script du lutin Horaire pour afficher l'heure sous la forme « 3:25:00 pm » (format 12 h) au lieu de « 15:25:00 » (format 24 h). Trouvez d'autres moyens d'améliorer cette application et implémentez-les.

Le jeu du Chasseur d'oiseaux

Chasseur dOiseaux .sb2

Construisons maintenant un jeu simple qui utilise la plupart des blocs que nous avons présentés dans ce chapitre. Le but du jeu sera de tirer sur deux oiseaux dans le ciel. L'interface est présentée à la figure 7-23.

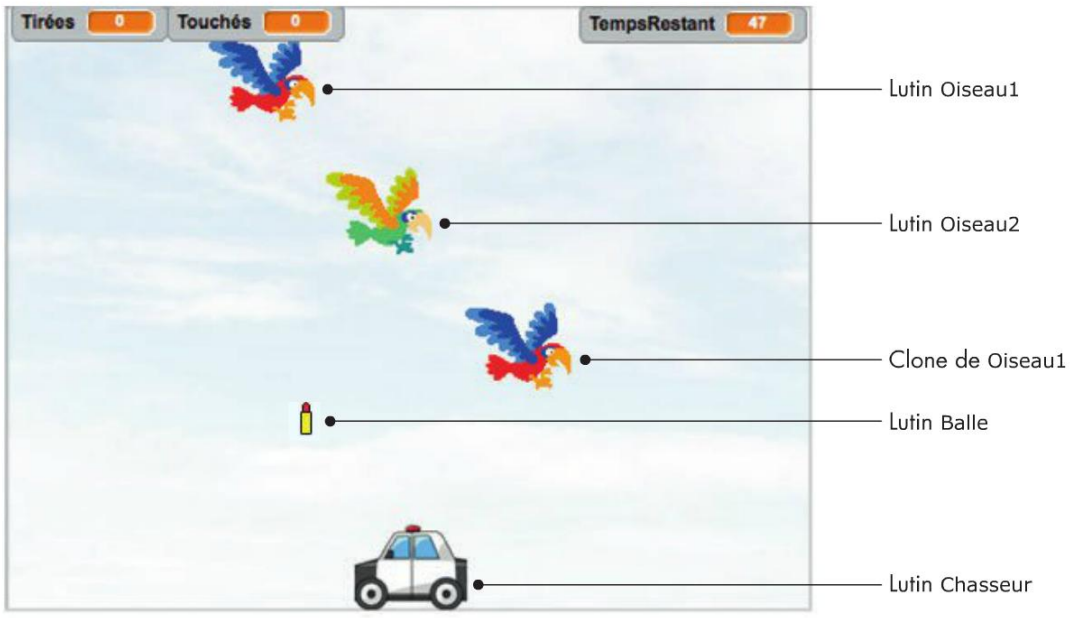


Figure 7-23 : Interface utilisateur du jeu de chasse d'oiseaux

Copyright © 2016 Eyrolles.

Comme vous le voyez, le jeu contient cinq lutins : Oiseau1, un clone de Oiseau1, Oiseau2, un chasseur et une balle. Le joueur déplace le chasseur horizontalement avec les flèches gauche et droite du clavier et il peut tirer en appuyant sur la barre d'espace. Si la balle atteint Oiseau1 ou son clone, le joueur gagne un point. Oiseau2 est une espèce en voie de disparition ; le joueur n'a donc pas le droit de lui tirer dessus. Si une balle touche ce lutin, le jeu se termine. Le joueur dispose d'une minute pour toucher le plus d'oiseaux possible. Chaque oiseau utilise deux costumes. En passant d'un costume à l'autre, les lutins semblent battre des ailes.

La scène possède deux arrière-plans nommés début et fin. L'arrière-plan début est présenté à la figure 7-23. L'arrière-plan fin est identique, avec en plus les mots *Game Over* affichés au centre de l'image. Les scripts de la scène sont présentés à la figure 7-24.

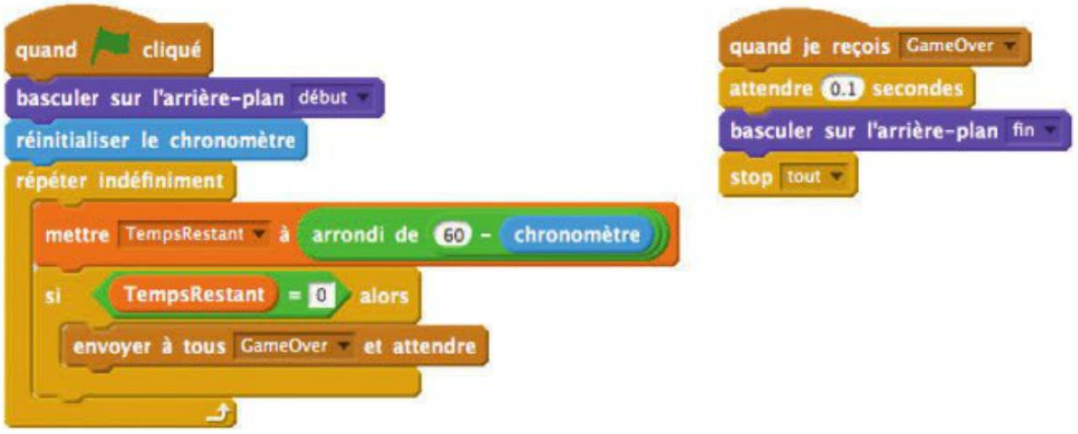


Figure 7-24 : Les scripts de la scène

Lorsqu'on clique sur le drapeau vert, la scène bascule vers l'arrière-plan début, réinitialise le chronomètre et démarre une boucle qui met à jour et vérifie le temps de jeu restant. Lorsque TempsRestant atteint 0 ou lorsque la scène reçoit le message GameOver, le script de gestion du message GameOver est exécuté. Ce script attend un court instant pour laisser les oiseaux se cacher, bascule vers l'arrière-plan fin et appelle **stop tout** pour arrêter tout script en cours d'exécution. Comme vous le verrez bientôt, le message GameOver sera également envoyé par le lutin Balle lorsqu'il touchera le lutin Oiseau2. Jetons à présent un œil au script du lutin Chasseur (figure 7-25).

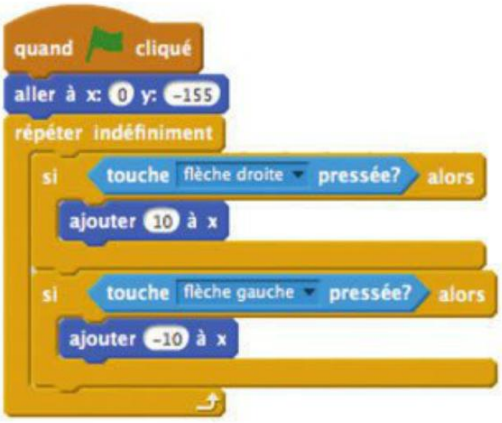


Figure 7-25 : Le script du lutin Chasseur

Copyright © 2016 Eyrolles.

Ce script démarre en positionnant le chasseur au milieu du bas de la scène. Il entre ensuite dans une boucle infinie qui détecte si l'une des flèches de gauche ou de droite est pressée et déplace le chasseur dans la direction correspondante. Passons maintenant aux scripts de Oiseau1 (figure 7-26).

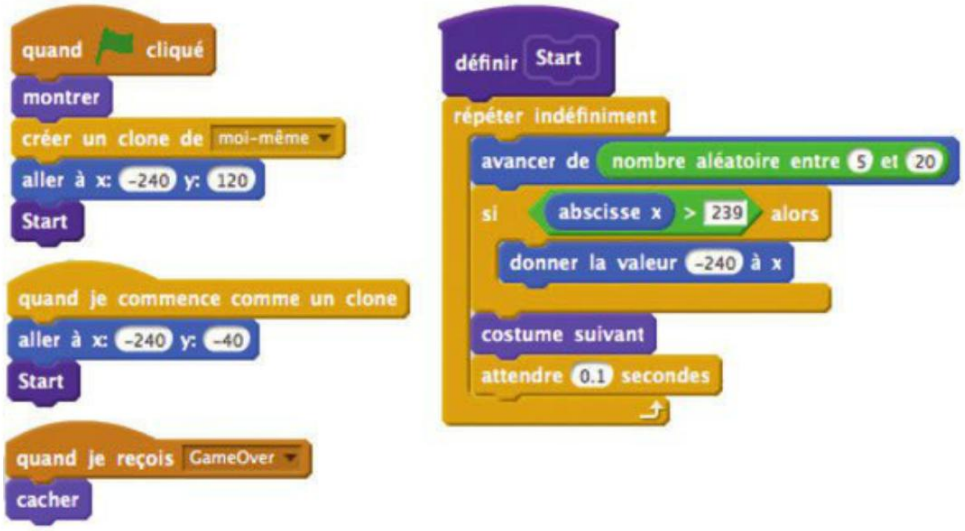


Figure 7-26 : Les scripts du lutin Oiseau1

Lorsque le jeu démarre, Oiseau1 crée un clone de lui-même, se déplace à gauche de la scène et fait appel à la procédure **Démarrer**. Le clone démarre également à gauche de la scène, mais avec une hauteur différente, puis appelle **Démarrer**. Cette procédure utilise une boucle **répéter indéfiniment** pour déplacer l'oiseau et son clone de gauche à droite sur la scène, en utilisant un nombre de pas aléatoire à chaque déplacement. Quand l'oiseau arrive au bord droit de la scène, il est déplacé au bord gauche, comme s'il faisait le tour et réapparaissait. Le dernier script cache les deux oiseaux lors de la réception du message GameOver.

Les scripts de Oiseau2 sont très similaires à ceux de Oiseau1. Lorsqu'on clique sur le drapeau vert, Oiseau2 se déplace vers le côté droit de la scène à une hauteur de 40 et exécute une boucle similaire à celle de la procédure **Démarrer** de la figure 7-26. L'oiseau se déplace de gauche à droite, en faisant le tour lorsqu'il atteint le bord droit de la scène. Oiseau2 répond également au message GameOver en se cachant.

Le joueur ne peut évidemment pas toucher d'oiseau en déplaçant simplement le chasseur et c'est là que le lutin Balle entre en jeu. Son script principal est présenté à la figure 7-27.

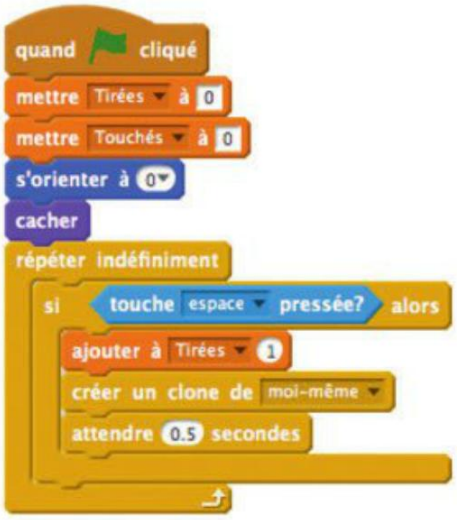


Figure 7-27 : Le script principal du lutin Balle

Lorsqu'on clique sur le drapeau vert, le script initialise les variables Tirées

Copyright © 2016 Eyrolles.

(le nombre de balles tirées) et Touchés (le nombre d'oiseaux touchés) à 0. Il oriente ensuite le lutin Balle vers le haut puis le cache. Après cela, il entre dans une boucle infinie pour vérifier en permanence l'état de la barre d'espace. Lorsqu'elle est pressée, le script incrémente la variable Tirées de 1 et crée un clone du lutin Balle pour déplacer la balle vers le haut, comme nous le verrons ensuite. Le script attend ensuite un certain temps pour empêcher le joueur de tirer une autre balle trop tôt. Nous sommes maintenant prêts à étudier le script de la balle clonée, présenté à la figure 7-28.

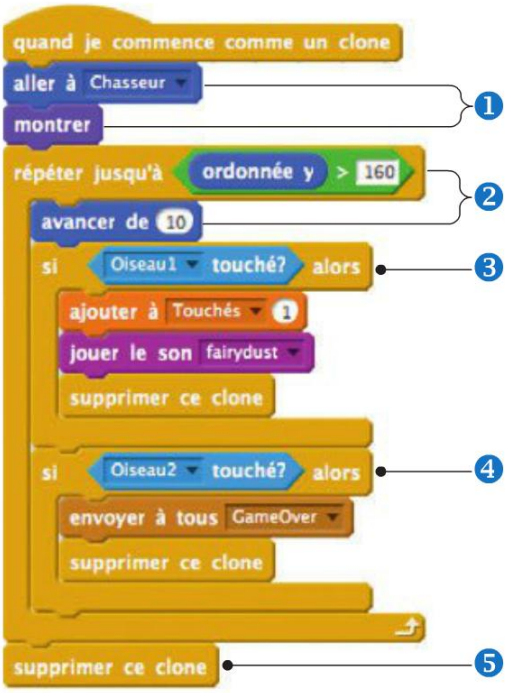


Figure 7-28 : Le script de gestion de démarrage d'un clone du lutin Balle

Tout d'abord, la balle est déplacée au centre du lutin Chasseur et est rendue visible ❶. Elle est ensuite déplacée vers le haut par incrémentations de 10 pas grâce au bloc **répéter jusqu'à** ❷. Si sa coordonnée *y* dépasse 160, alors le lutin Balle aura atteint le bord supérieur de la scène sans avoir touché d'oiseau. Dans ce cas, le programme sort de la boucle **répéter jusqu'à** ❺ et le clone est supprimé. Cependant, le programme vérifie si la balle a touché un oiseau à chaque déplacement de la balle. Si elle touche Oiseau1 (ou son clone) ❸, le script incrémente la variable Touchés de 1 et joue un son pour rendre le jeu plus palpitant. En revanche, si la balle touche Oiseau2 ❹, le script envoie le message GameOver pour signaler la fin du jeu. Dans les deux cas, le clone est supprimé puisqu'il n'a plus rien à faire.

Le jeu est à présent entièrement fonctionnel, mais vous pourriez y ajouter beaucoup de fonctions. Voici deux suggestions :

- Donner au joueur un nombre de balles limité et compter le score par rapport au nombre de tirs ratés.
- Ajouter des oiseaux et les faire se déplacer à des vitesses différentes. Récompenser le joueur avec plus de points lorsqu'un oiseau plus rapide est touché.

FAITES LE TEST 7-8

Ouvrez le jeu et jouez-y pour voir comment il fonctionne. Modifiez-le avec certaines des améliorations proposées, ou trouvez vous-même quelques fonctionnalités à apporter et implémentez-les !

Copyright © 2016 Eyrolles.

Simulation de chute libre

ChuteLibre.sb2

Dans cette section, je présenterai une application qui simule le mouvement d'un objet tombant. En ignorant l'effet de flottaison et de résistance de l'air, lorsqu'un objet au repos est lâché d'une certaine hauteur, la distance d (en mètres) parcourue par l'objet tombant pendant un temps t (en secondes) est donnée par $d = ½\ gt^2$, avec $g = 9.8\ \text{m/s}^2$ l'accélération gravitationnelle. Le but de cette simulation est de montrer à intervalles de 0,5 s la position de l'objet jusqu'à ce qu'il atteigne le sol. L'interface est présentée à la figure 7-29.

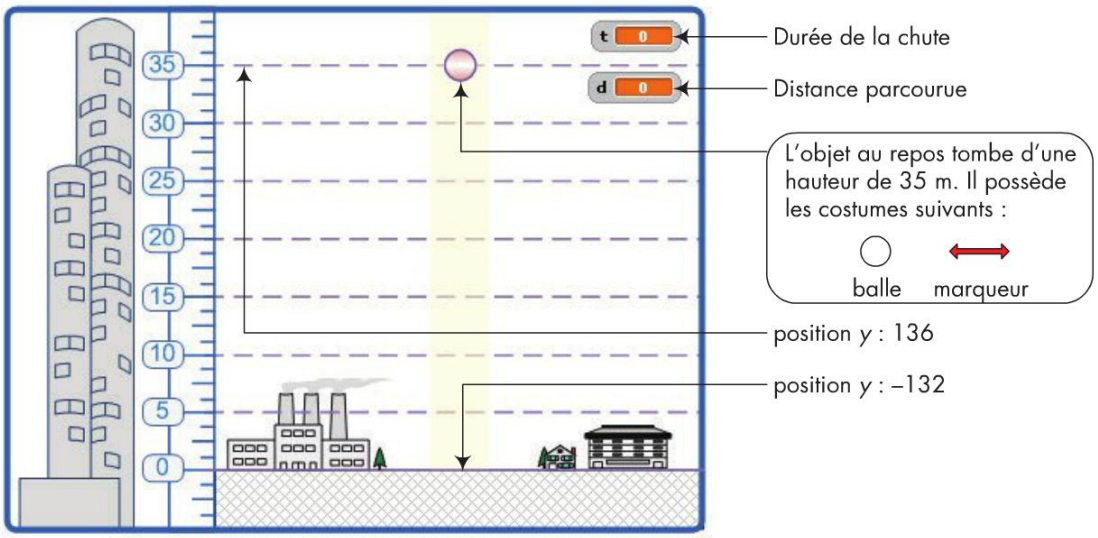


Figure 7-29 : Interface utilisateur de la simulation de chute libre

Un objet au repos (la balle sur la figure) va tomber d'une hauteur de 35 m. Une simple substitution de la formule précédente montre que l'objet atteindra le sol après $t = \sqrt{(2 \times 35) / 9{,}8} = 2{,}67$ s. L'application possède un lutin (appelé Balle) doté de deux costumes présentés sur la figure. Lorsqu'il est temps de montrer la position de la balle qui tombe, le lutin passe momentanément au costume marqueur, l'estampille et revient au costume balle.

La simulation démarre lorsqu'on clique sur le drapeau vert. Le lutin Balle démarre alors le script présenté à la figure 7-30.

Pendant l'initialisation ❶, le lutin se déplace à sa position initiale, passe au costume balle, efface sa bulle de l'exécution précédente et efface ses précédentes traces de la scène. Il initialise ensuite le compteur à 0. La variable t représente la durée de la chute et compteur garde une trace du nombre de répétitions de la boucle.

Le script entre ensuite dans une boucle infinie ❷ pour calculer les paramètres de la simulation à différents intervalles de temps. Il effectue les calculs et met à jour la position de la balle toutes les 0,05 s ❸ pour s'assurer de son déplacement continu : la valeur de la variable t de temps est mise à jour et la distance d que la balle a parcourue est calculée. La valeur de la variable compteur est incrémentée de 1.

Copyright © 2016 Eyrolles.

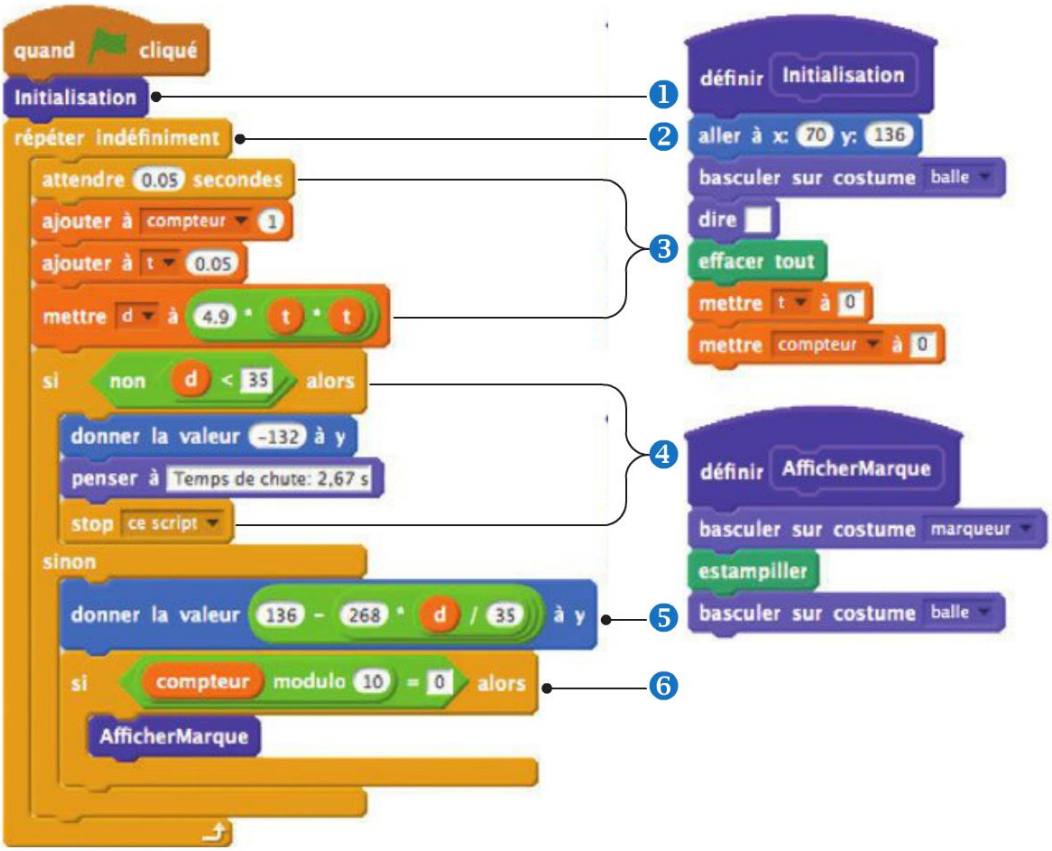


Figure 7-30 : Script du lutin Balle *de la simulation de chute libre*

Si la balle atteint le sol (ce qui arrive à d ≥ 35), le script affecte à la position *y* de la balle la position *y* du sol, affiche la durée actuelle du trajet et met fin à la simulation ❹.

Sinon, le script règle la position verticale de la balle selon la distance de chute parcourue ❺. Puisqu'une hauteur de 35 m correspond à 268 pixels sur la scène (figure 7-29), une distance de *d* mètres correspond à 268 × (d / 35). La position *y* finale est établie en soustrayant ce nombre à la position *y* initiale, qui est 136.

Puisque la durée d'une itération est de 0,05 s, cela prendra 10 itérations pour atteindre 0,5 s. Donc, lorsque le compteur passe à 10, 20, 30 et ainsi de suite, le lutin Balle bascule vers (et estampille) le costume marqueur pour montrer la position de la balle tombante à ces instants ❻.

La figure 7-31 illustre le résultat de cette simulation. Vous remarquerez la façon dont la distance parcourue augmente à chaque intervalle de temps pendant la chute. À cause de la gravité, la balle accélère – sa vitesse augmente – à une allure de 9,8 m/s^2.

FAITES LE TEST 7-9

Ouvrez et exécutez l'application pour comprendre son fonctionnement. Essayez de convertir cette simulation en un jeu où le joueur lâche la balle pour toucher un objet mobile sur le sol. Vous pouvez ajouter un score, changer la vitesse de la cible, ou même situer l'action sur une autre planète (en changeant l'accélération gravitationnelle).

Copyright © 2016 Eyrolles.

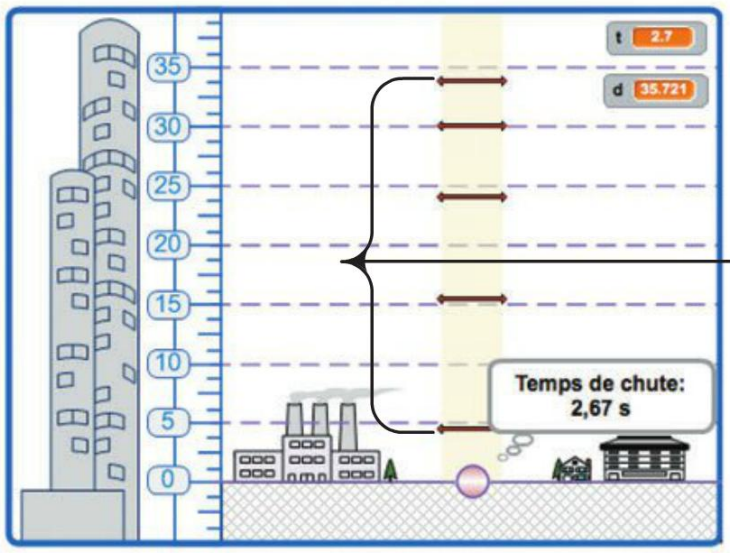


Figure 7-31 : Sortie de la simulation de chute libre

Simulation du mouvement d'un projectile

Projectile.sb2

Considérez un boulet tiré avec une certaine vitesse initiale (v_0) d'un canon orienté à un angle θ par rapport à l'axe horizontal. Vous pouvez analyser la trajectoire du boulet en résolvant les équations liant sa position et les composantes du vecteur vitesse à différents instants. La composante horizontale reste constante, mais la composante verticale est affectée par la gravité. Le trajet résultant est une parabole. Examinons les équations qui gouvernent le mouvement du projectile (en négligeant la résistance de l'air).

L'origine de notre système de coordonnées est le point de départ du boulet. La coordonnée x du boulet à chaque instant t est donc donnée par $x(t) = v_{0x}t$ et la coordonnée y par $y(t) = v_{0y}t - (0,5)gt^2$, avec $v_{0x} = v_0\cos\theta$ et $v_{0y} = v_0\sin\theta$ les composantes x et y de v_0 et $g = 9,8$ m/s^2 l'accélération gravitationnelle. Nous pouvons calculer grâce à ces équations la durée totale du trajet, la hauteur maximale et la distance horizontale parcourue par le boulet (figure 7-32).

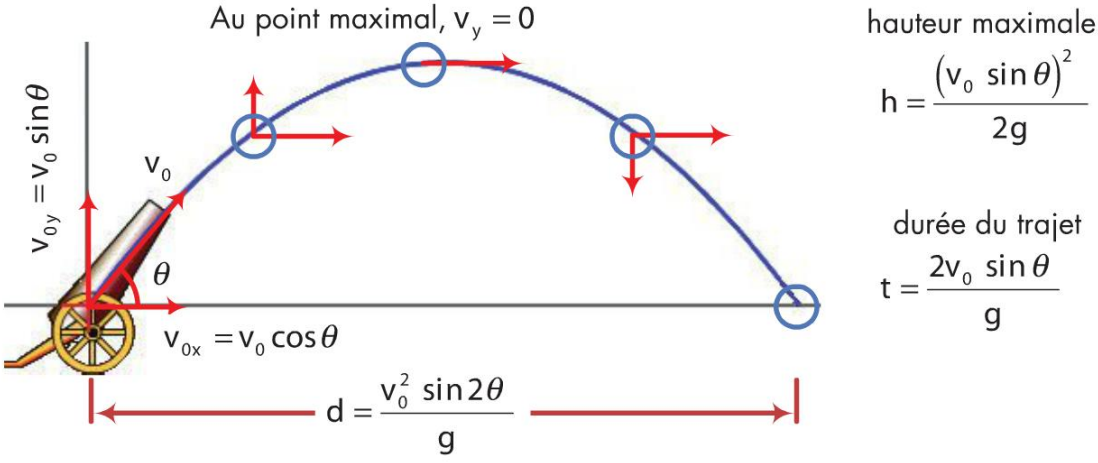


Figure 7-32 : Trajectoire parabolique du boulet

Copyright © 2016 Eyrolles.

Nous avons toutes les informations nécessaires à la réalisation d'une simulation réaliste du mouvement du boulet. Créons un programme Scratch qui montre ces règles physiques en action pour approfondir notre compréhension des trajectoires. L'interface utilisateur de cette simulation est présentée à la figure 7-33.

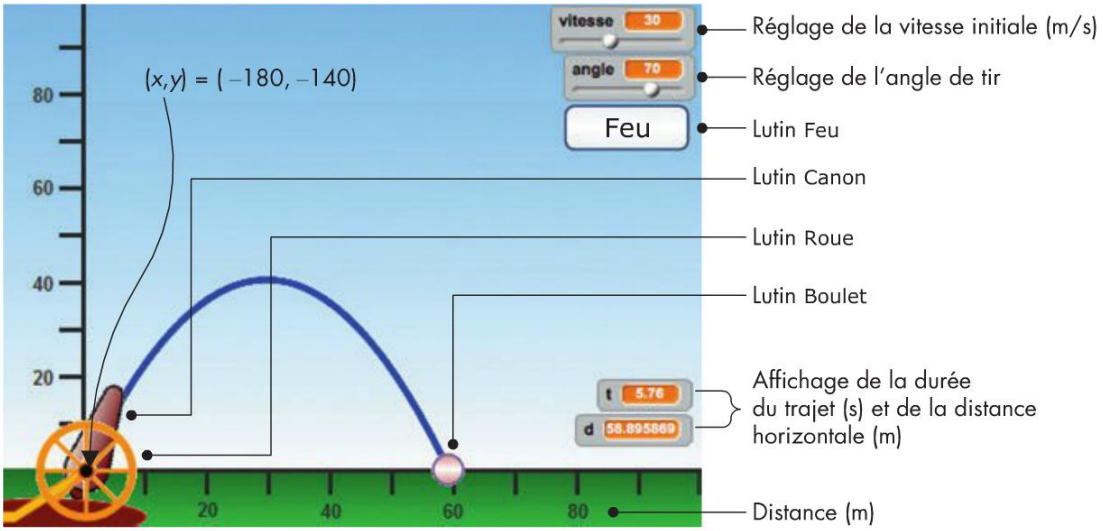


Figure 7-33 : Interface utilisateur du simulateur de mouvement d'un projectile

Comme vous pouvez le voir, l'application contient quatre lutins : Roue fournit un axe de rotation au canon, Canon pivote en fonction du potentiomètre angle et fournit une indication visuelle de l'angle de tir, Feu est un bouton sur lequel l'utilisateur cliquera pour lancer le boulet et Boulet contient le script principal calculant les coordonnées du projectile et traçant sa trajectoire. L'utilisateur règle l'angle de tir et la vitesse initiale grâce aux deux potentiomètres, puis clique sur le bouton Feu. Le boulet part du point (-180,-140) de la scène et dessine une trajectoire parabolique correspondant aux paramètres spécifiés. Les deux moniteurs en bas à droite de la scène affichent la durée du trajet et la distance horizontale parcourue en temps réel.

La simulation démarre lorsqu'on clique sur le drapeau vert. Les scripts du lutin Canon (non présentés ici) orientent le canon dans la direction spécifiée par le potentiomètre angle. L'utilisateur peut également régler l'angle en cliquant sur le canon et en le faisant glisser. Lorsque l'utilisateur clique sur le bouton Feu, un message Feu est envoyé, que le lutin Boulet reçoit et traite avec le script de la figure 7-34.

Pour se préparer à tirer ❶, le lutin Boulet se place au premier plan (devant le canon et la roue) et se positionne au point de lancement. Il met ensuite son stylo en position d'écriture et efface tout précédent tracé de la scène. Le script calcule ensuite les composantes verticale (ou *x*) et horizontale (ou *y*) de la vitesse initiale (nommées vx et vy respectivement) et initialise la variable temporelle (t) à 0.

Copyright © 2016 Eyrolles.

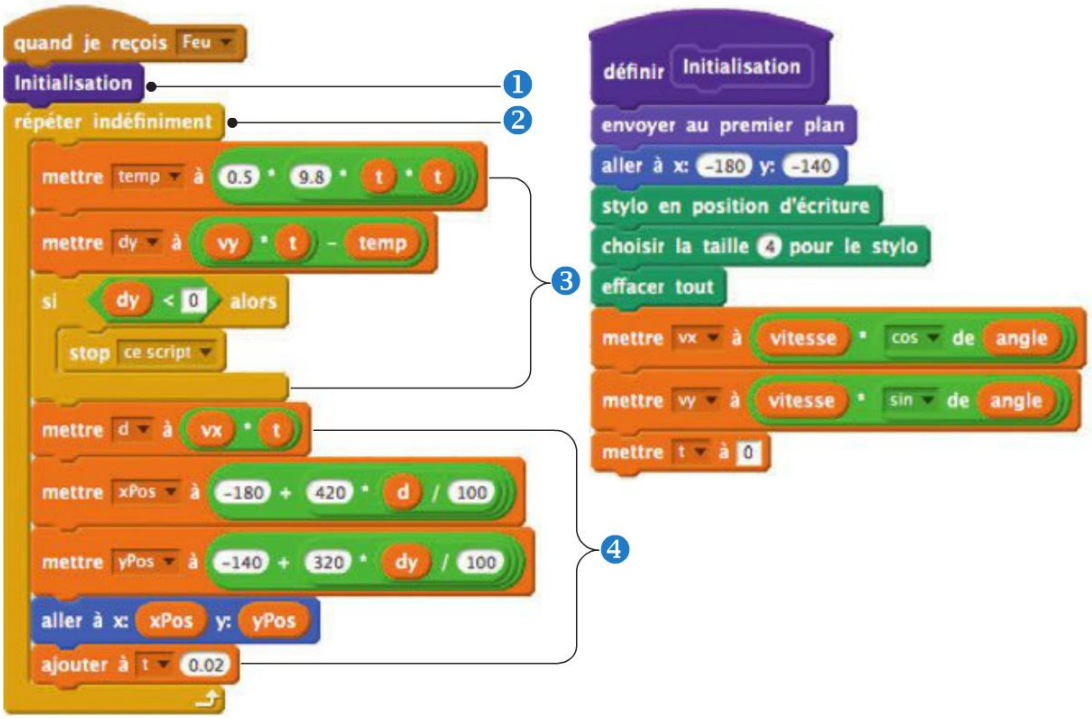


Figure 7-34 : Script du lutin Boulet

Le script entre ensuite dans une boucle infinie ❷, qui calcule et met à jour la position du boulet toutes les 0,02 s. Tout d'abord, la distance verticale (`dy`) du lutin est calculée ❸. Si la valeur calculée est négative, alors la balle aura atteint le sol. À cet instant, la commande **stop ce script** est appelée pour mettre fin à la simulation.

Si `dy` n'est pas négatif, la distance horizontale (`d`) est calculée ❹. Les deux distances (`dy` et `d`) sont ensuite redimensionnées par rapport à l'arrière-plan de la scène. La direction verticale contient 320 pas (de -140 à 180) qui correspondent à 100 m et la position horizontale contient 420 pas (de -180 à 240) qui correspondent à 100 m. Cela signifie qu'une distance verticale de `dy` mètres est équivalente à `320 × dy / 100` pas et qu'une distance horizontale de `d` mètres est équivalente à `420 × d / 100` pas. Les coordonnées *x* et *y* sont mises à jour et le boulet est déplacé à sa position actuelle sur sa trajectoire. La variable temporelle (`t`) est ensuite incrémentée d'une faible valeur (0,02 dans ce cas) et la boucle est répétée pour calculer la prochaine position du boulet.

Par exemple, si le boulet est projeté avec un angle de tir de 70° et une vitesse initiale de 30 m/s, comme sur la figure 7-33, alors la durée totale du trajet sera de 5,76 s et la distance horizontale de 59 m. En regardant les moniteurs de la figure 7-33, nous voyons que notre simulation est très précise. Nous pourrions l'améliorer en mettant à jour nos calculs plus fréquemment (par exemple toutes les 0,01 s au lieu de 0,02 s), mais cela la ralentirait. Il est important d'ajuster ce paramètre pour adopter un bon compromis entre vitesse et précision.

Copyright © 2016 Eyrolles.

FAITES LE TEST 7-10

Ouvrez l'application et exécutez-la pour voir comment elle fonctionne. Essayez ensuite d'en faire un jeu. Vous pourriez par exemple montrer un objet à une hauteur aléatoire sur le bord droit de la scène et demander au joueur d'essayer de le toucher. Si le joueur rate la cible, le jeu peut donner des indices sur la façon d'ajuster l'angle de tir et la vitesse initiale.

Autres applications

MatchThatAmount.sb2

Les applications bonus de ce livre contiennent trois jeux supplémentaires que vous pourrez étudier par vous-même, avec une explication complète (en anglais) de chaque script. Le premier d'entre eux est un jeu éducatif testant les capacités de calcul mental d'élèves de primaire. Un certain montant d'argent est affiché en centimes et le jeu demande au joueur de trouver le plus petit nombre de pièces nécessaire pour obtenir ce montant.

Orbit.sb2

La seconde application est une simulation de mouvement planétaire d'un système solaire simple qui contient un soleil et une unique planète.

MoleculesInMotion.sb2

La troisième application est également une simulation, démontrant la dynamique du mouvement d'une molécule de gaz lorsqu'elle entre en collision avec les parois d'un récipient.

Ouvrez ces applications, exécutez-les et lisez mes explications pour comprendre comment elles fonctionnent. Si vous sentez l'inspiration vous envahir, essayez de modifier les scripts pour leur faire accomplir de nouvelles choses !

Résumé

Nous avons vu dans ce chapitre différentes manières de répéter des commandes dans Scratch. Nous avons d'abord étudié les différents blocs de boucle et avons expliqué les termes techniques associés. Nous avons ensuite exposé la différence entre les boucles comptées (ou définies) et les boucles conditionnelles (ou indéfinies). Nous avons étudié le bloc **répéter jusqu'à** ainsi que la structure **répéter indéfiniment / si** puis nous les avons utilisés dans plusieurs exemples. Je vous ai également expliqué le fonctionnement des commandes **stop** de Scratch et la façon de s'en servir pour mettre fin à des boucles infinies ou à des procédures. À partir de là, nous avons étudié comment utiliser des boucles pour valider les entrées utilisateur.

Vous avez ensuite appris à employer des compteurs, d'une part pour garder une trace du nombre des itérations réalisées par une boucle et, d'autre part, dans des boucles imbriquées pour créer des itérations dans deux dimensions ou plus. Nous avons ensuite traité de la récursivité (une procédure s'appelant elle-même), un autre moyen de coder des répétitions. Dans la dernière section, nous avons développé plusieurs applications qui ont assemblé ces nouveaux concepts pour créer des programmes pratiques.

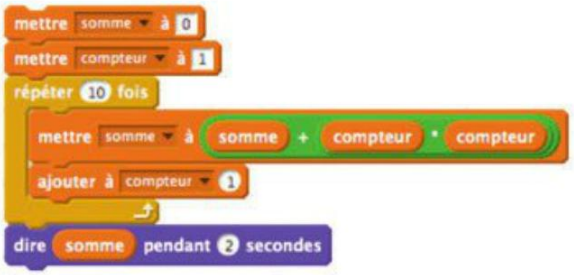
Copyright © 2016 Eyrolles.

Le prochain chapitre ira plus loin dans les concepts que vous avez vus ici et vous enseignera comment utiliser des compteurs et des boucles pour traiter des chaînes de caractères et créer différents types de programmes intéressants, tels qu'un convertisseur binaire/décimal, un jeu de pendu et un programme de mathématiques pour apprendre les fractions.

Si vous désirez aller un peu plus loin dans les concepts évoqués dans ce chapitre, je vous suggère d'essayer de résoudre les problèmes suivants.

Problèmes

1. Créez une boucle de validation d'entrées qui n'accepte que les nombres compris entre 1 et 10.
2. Écrivez un script qui demande à l'utilisateur : « Voulez-vous vraiment quitter [O, N] ? » Le script vérifiera l'entrée utilisateur et n'acceptera que les lettres O et N comme entrées valides.
3. Écrivez un programme qui calcule et affiche la somme de tous les entiers allant de 1 à 20.
4. Écrivez un programme qui calcule et affiche la somme de tous les entiers pairs allant de 1 à 20.
5. Écrivez un programme qui affiche les 10 premiers nombres de la séquence suivante (en utilisant la commande **dire**) : 5, 9, 13, 17, 21 …

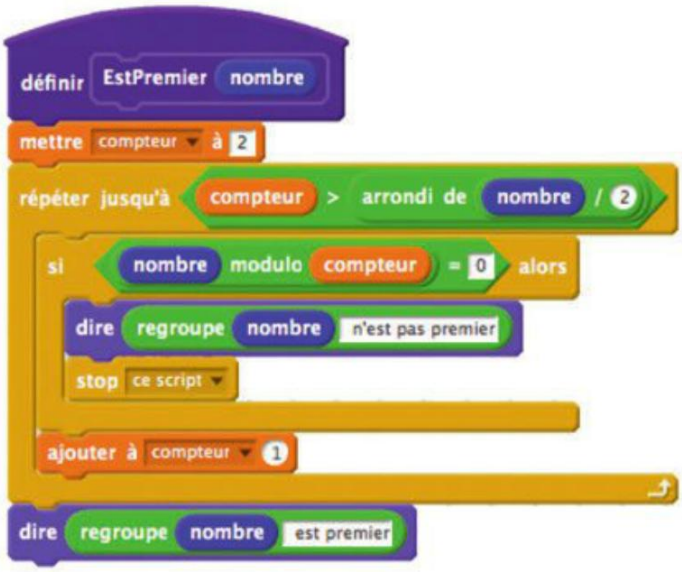


6. Que fait le script ci-après ? Implémentez-le et exécutez-le pour vérifier votre réponse.
7. Si le reste de la division d'un entier x par un autre entier y est 0, alors y est un facteur de x. Par exemple, 1, 2, 4 et 8 sont des facteurs de 8. Le script ci-après trouve et affiche tous les facteurs d'un nombre donné (autres que le nombre lui-même). Etudiez ce script et expliquez comment il fonctionne. Quelles en seront les sorties pour les entrées 125, 324 et 419 ?

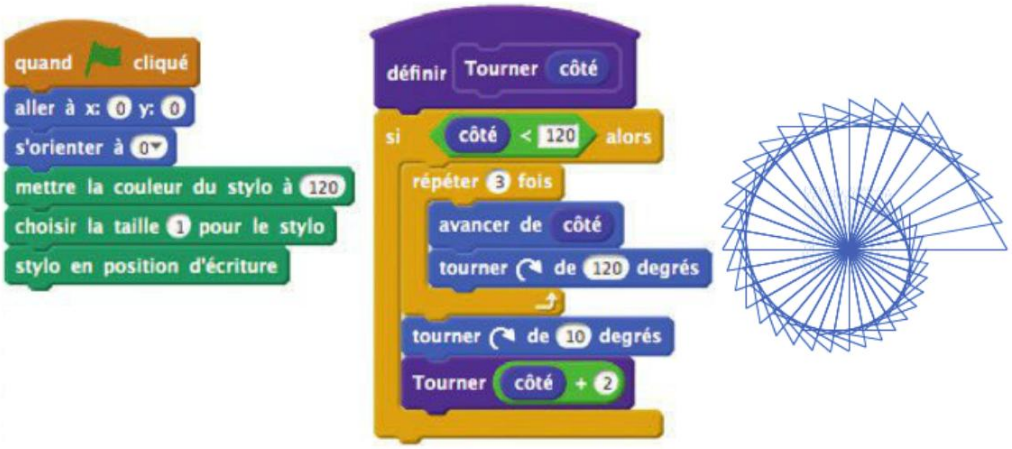


Copyright © 2016 Eyrolles.

8. Un entier est dit premier s'il n'est divisible que par 1 et lui-même. Par exemple, 2, 3, 5, 7 et 11 sont des nombres premiers, alors que 4, 6 et 8 ne le sont pas. La procédure qui suit teste si un nombre est premier ou non. Étudiez-la et expliquez comment elle fonctionne. Quelles seront les sorties du script pour les entrées 127, 327 et 523 ?



9. La procédure du problème 8 vérifie tous les entiers jusqu'à la moitié de la valeur d'entrée. Il serait en fait suffisant de ne pas dépasser la valeur de sa racine carrée. Modifiez la procédure et vérifiez si elle donne toujours les mêmes réponses.
10. La séquence de nombres 0, 1, 1, 2, 3, 5, 8, 13, 21, 34… est appelée série de Fibonacci. Les deux premiers nombres de la série sont 0 et 1. Chaque nouveau nombre est obtenu par la somme des deux précédents. Écrivez un programme qui calcule le *n*e terme d'une série de Fibonnacci, où *n* est entré par l'utilisateur.
11. Considérez le programme suivant ainsi que sa sortie. Recopiez-le et exécutez-le pour voir comment il fonctionne. Modifiez l'angle de rotation ainsi que l'argument de l'appel récursif (à **côté + 1** ou **côté + 3**, par exemple) pour découvrir ce que vous pouvez créer d'autre.



Copyright © 2016 Eyrolles.

8

TRAITEMENT DES CHAÎNES DE CARACTÈRES

Une chaîne de caractères est considérée comme une variable unique, comportant plusieurs caractères. Vous pouvez écrire des programmes pour combiner, comparer, trier, encrypter, ou plus généralement manipuler des chaînes de caractères. Voici ce que vous apprendrez dans ce chapitre :

- comment Scratch stocke les chaînes de caractères ;
- comment manipuler les chaînes avec les blocs disponibles ;
- plusieurs techniques de traitement de chaînes ;
- comment écrire des programmes intéressants qui manipulent des chaînes.

Nous commencerons par étudier en détail le type de données « chaîne de caractères » et nous écrirons des procédures gérant et manipulant ces données. Ces procédures supprimeront ou remplaceront des caractères,

Copyright © 2016 Eyrolles.

inséreront ou extrairont des sous-chaînes, changeront l'ordre des caractères de manière aléatoire. Ensuite, nous utiliserons ces procédures et appliquerons ces techniques pour créer des applications pratiques et amusantes.

Retour sur les chaînes de caractères

Une chaîne de caractères est simplement une séquence ordonnée de caractères. Ces derniers peuvent inclure des lettres (majuscules et minuscules), des chiffres et d'autres symboles disponibles sur votre clavier (+, –, &, @, etc.). Dans vos programmes, les chaînes stockent des noms, des adresses, des numéros de téléphone, des titres de livres et bien d'autres choses.

Dans Scratch, les caractères d'une chaîne sont stockés de manière séquentielle. Par exemple, si vous avez une variable appelée nom, utiliser la commande **mettre nom à Julie** stockera les caractères comme le montre la figure 8-1.

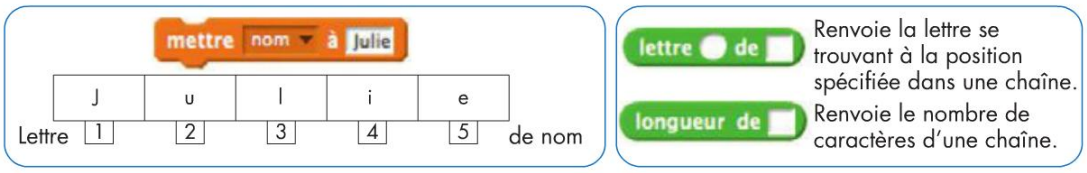


Figure 8-1 : Une chaîne est stockée comme une séquence de caractères.

Vous pouvez accéder à chaque caractère d'une chaine individuellement avec l'opérateur **lettre de**. Par exemple, le bloc **lettre 1 de nom** revoie la lettre K, et **lettre 5 de nom** renvoie la lettre n. Scratch fournit également l'opérateur **longueur de**, qui renvoie le nombre de caractères contenus dans une chaine. Si vous utilisez ces deux opérateurs avec un bloc **répéter**, vous pouvez compter des caractères, examiner plusieurs caractères, et faire bien d'autres choses utiles que je vous montrerai dans la section suivante.

Compter les caractères d'un certain type dans une chaîne

ComptageDeVoyelles.sb2

Notre premier exemple de script (figure 8-2) demande à l'utilisateur de rentrer une chaîne, puis compte et affiche le nombre de voyelles qui s'y trouvent.

Copyright © 2016 Eyrolles.

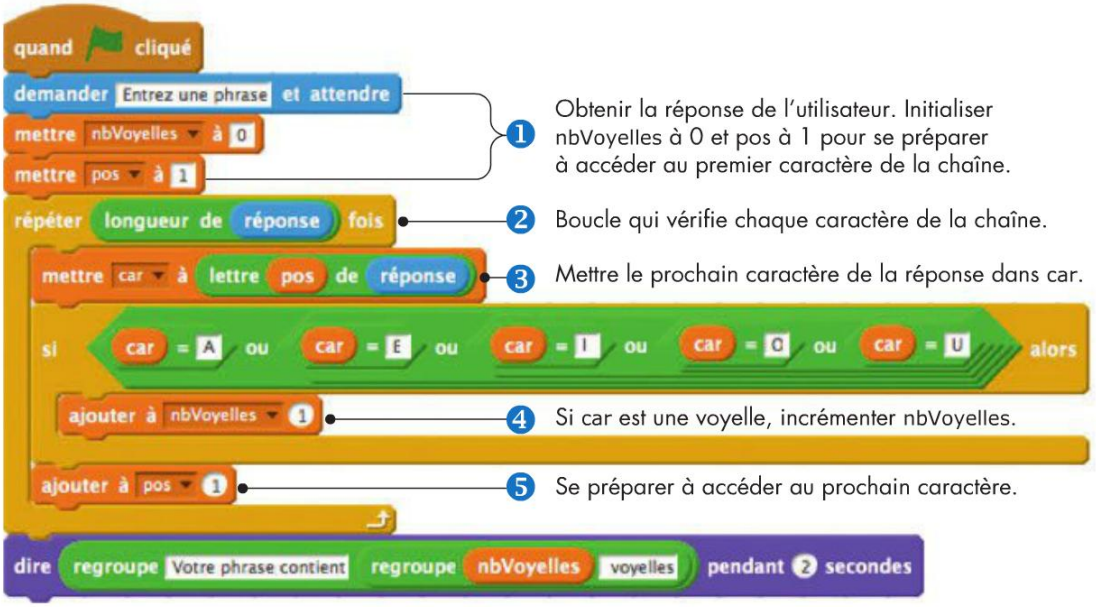


Figure 8-2 : Programme de comptage de voyelles

Le programme vérifie un à un les caractères de la chaîne d'entrée pour savoir s'il s'agit d'une voyelle. Si c'est le cas, il incrémente de 1 une variable nommée nbVoyelles. Le script utilise une variable nommée pos (pour *position*) afin de connaître à tout moment la position du caractère observé. Étudions ce script en détail.

Tout d'abord, le script demande à l'utilisateur de rentrer une phrase ❶. Scratch enregistre automatiquement cette chaîne dans la variable intégrée réponse. Il initialise ensuite la variable nbVoyelles à 0 (puisqu'il n'a pas encore vu de voyelle) et la variable pos à 1 pour accéder au premier caractère de la chaîne.

Ensuite, une boucle **répéter** ❷ vérifie chaque caractère. L'opérateur **longueur de** renvoie le nombre des caractères contenus dans la chaîne, qui correspond au nombre d'itérations que la boucle devra effectuer.

À chaque itération, la boucle utilise car (pour caractère) pour contrôler un caractère de la chaîne d'entrée ❸. Le premier caractère de **réponse** est affecté à car dans la première itération de la boucle, puis le deuxième caractère de **réponse** lui est affecté lors de la deuxième itération et ainsi de suite, jusqu'à ce que la boucle atteigne le dernier caractère. La variable pos est utilisée pour accéder au caractère désiré.

Le bloc **si** vérifie ensuite si le caractère examiné est une voyelle ❹. Si c'est le cas (majuscule ou minuscule), nbVoyelles est incrémenté de 1.

Après avoir contrôlé un caractère, la boucle incrémente pos de 1 ❺ et redémarre pour vérifier le suivant. Quand toutes les lettres de la chaîne d'entrée ont été contrôlées, la boucle se termine et le programme affiche le nombre de voyelles qu'il a comptées grâce au bloc **dire**.

Les techniques utilisées dans cet exemple seront appliquées à de nombreuses reprises dans le reste de ce chapitre. Ouvrez le script *ComptageDeVoyelles.sb2*, exécutez-le plusieurs fois et assurez vous de l'avoir entièrement compris.

Copyright © 2016 Eyrolles.

Comparer des chaînes de caractères

Palindrome.sb2

Notre deuxième exemple vérifie si un entier entré est un palindrome. Un *palindrome* est un nombre ou une chaîne de caractères dont la lecture à l'endroit ou à l'envers donne un résultat identique. Par exemple, le mot *ressasser* et *Bob* ainsi que les nombres 1234321 et 1122332211 sont des palindromes. Pour illustrer notre algorithme de vérification de palindrome, prenons le chiffre 12344321 comme entrée (figure 8-3).

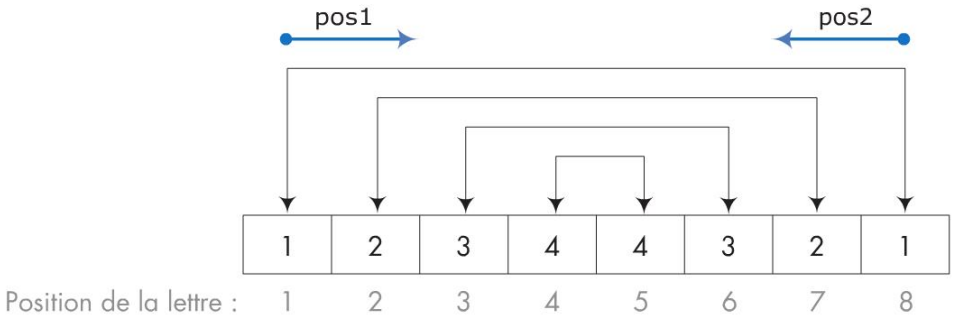


Figure 8-3 : Utilisation de deux variables pour vérifier si un nombre est un palindrome ou non

Pour vérifier si un nombre est un palindrome, nous devons comparer le premier et le huitième chiffre, le deuxième et le septième, le troisième et le sixième, le quatrième et le cinquième. Si l'une de ces comparaisons renvoie un résultat faux (signifiant que les chiffres comparés ne sont pas égaux), alors le nombre n'est pas un palindrome. Un programme qui implémente cet algorithme est présenté à la figure 8-4.

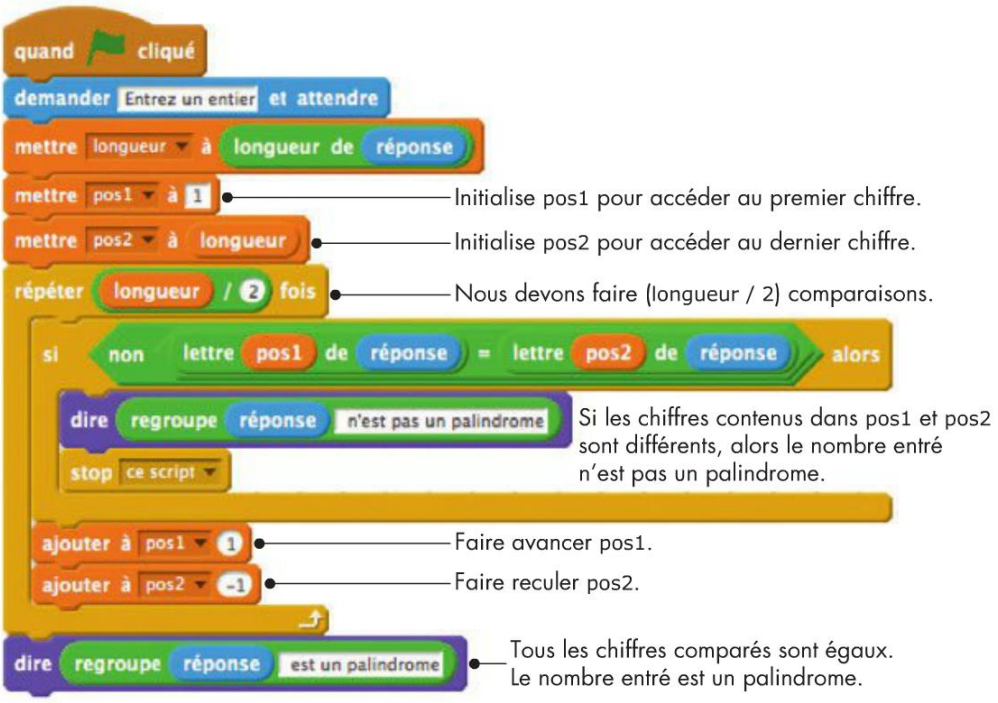


Figure 8-4 : Ce programme teste si un entier entré par l'utilisateur est un palindrome ou non.

Copyright © 2016 Eyrolles.

Le script accède aux chiffres à comparer grâce aux variables `pos1` et `pos2`. La variable `pos1` démarre au premier chiffre puis avance, tandis que `pos2` démarre au dernier chiffre puis recule. Le nombre maximal de comparaisons requises correspond à la moitié du nombre de chiffres de la chaîne. Avec 12344321, nous avons besoin de quatre comparaisons au maximum puisqu'il contient huit chiffres. La même logique s'applique aux nombres comportant un nombre impair de chiffres, puisque le chiffre du milieu n'a pas besoin d'être comparé. Une fois que le programme a déterminé si le nombre entré par l'utilisateur est un palindrome ou non, il affiche le message avec le résultat.

Palindrome.sb2

FAITES LE TEST 8-1

Ouvrez l'application *palindrome.sb2* et exécutez-la pour comprendre comment elle fonctionne. À cause de la manière dont Scratch gère les répétitions d'un nombre décimal de fois, si l'utilisateur rentre un nombre impair de chiffres, le script effectuera une comparaison en plus entre les deux chiffres entourant celui du milieu. Essayez d'améliorer le code pour effectuer le nombre correct de répétitions lorsque le nombre entré possède un nombre impair de chiffres.

Dans la section suivante, nous étudierons quelques-unes des opérations les plus courantes sur les chaînes de caractères et nous aborderons quelques stratégies pour écrire des procédures de manipulation de chaînes de caractères dans Scratch.

Exemples de manipulation de chaînes de caractères

L'opérateur **lettre de** ne lit qu'un seul caractère d'une chaîne. Si vous voulez insérer ou supprimer des caractères, vous devrez faire le travail vous-même.

Dans Scratch, vous ne pouvez pas modifier un caractère dans une chaîne, donc le seul moyen de le faire est de créer une nouvelle chaîne. Par exemple, si vous voulez mettre la première lettre de « jacques » en majuscule, vous devrez créer une nouvelle chaîne contenant la lettre *J* suivie du reste des caractères, *acques*. L'idée est d'utiliser l'opérateur **lettre de** pour lire les caractères de la chaîne originale et les ajouter comme désirés à une nouvelle chaîne, grâce à l'opérateur **regroupe**.

Dans cette section, nous développerons des applications simples démontrant des techniques courantes de manipulation de chaînes de caractères.

Copyright © 2016 Eyrolles.

Angagelille ecretsille

Louchébem.sb2

Et si nos lutins pouvaient parler un langage secret ? Dans cette section, nous leur apprendrons un langage codé appelé *louchébem*. Les règles de ce langage sont simples. Pour convertir un mot en louchébem, il faut déplacer la première lettre à la fin et y ajouter les lettres *ille*. Ainsi, le mot *parler* deviendrait *arlerpille*, *drôle* deviendrait *rôledille*, etc. Maintenant que vous connaissez les règles, pouvez-vous déchiffrer ce que signifie le titre de cette section ?

La stratégie que nous utiliserons pour convertir un mot en louchébem est illustrée à la figure 8-5 avec le mot *scratch*.

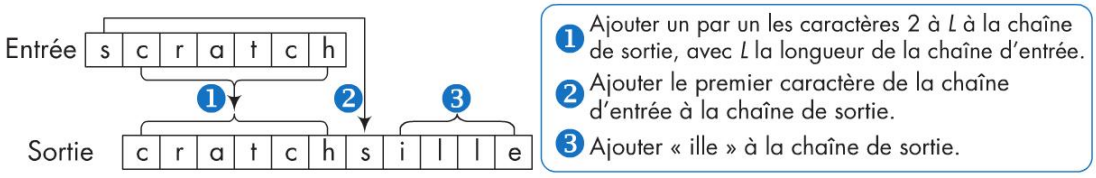


Figure 8-5 : Comment traduire un mot français en louchébem

Tout d'abord, nous ajoutons un par un tous les caractères de la chaîne d'entrée (sauf le premier) à la chaîne de sortie ❶. Nous y ajoutons ensuite la première lettre de la chaîne d'entrée ❷, suivie de *ille* ❸. La procédure **Louchébem** qui implémente ceci est présentée à la figure 8-6.

Cette procédure utilise trois variables pour créer notre mot codé. La variable motDeSortie stocke la chaîne de caractères de sortie pendant son assemblage. Un compteur appelé pos (pour position) indique au script quel caractère de la chaîne originale doit être ajouté à motDeSortie. Enfin, une variable nommé car stocke un caractère de la chaîne d'entrée. La procédure prend mot comme paramètre d'entrée, représentant le mot que vous désirez traduire en louchébem.

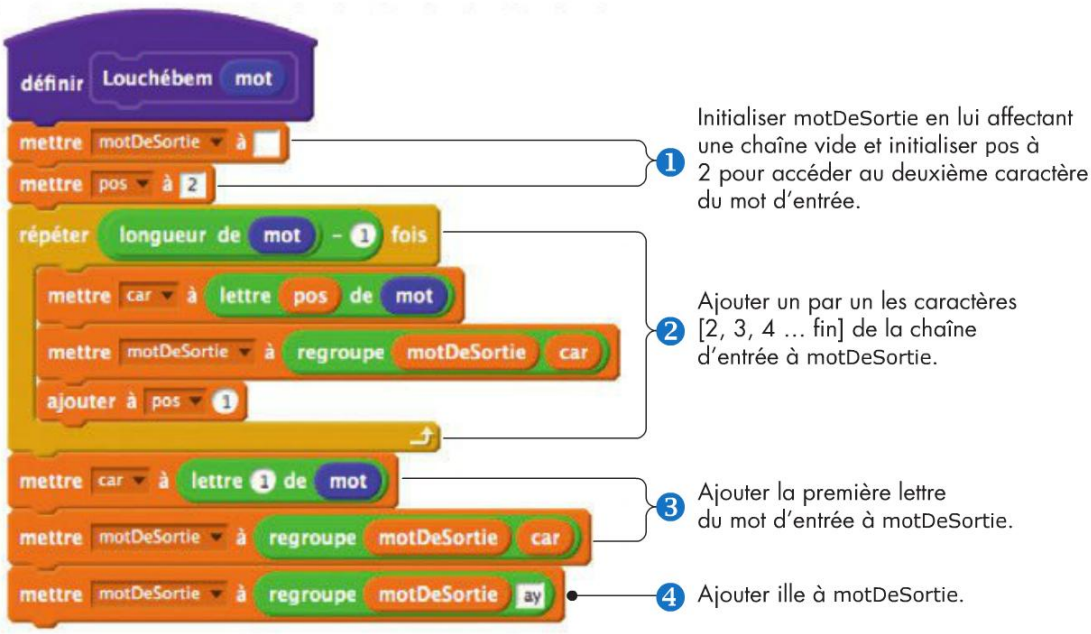


Figure 8-6 : La procédure **Louchébem**

Copyright © 2016 Eyrolles.

Tout d'abord, la procédure crée une chaîne de caractères vide pour motDeSortie et initialise pos à 2 ❶. Une *chaîne de caractères vide* ne contient aucun caractère ; elle a une longueur de 0. La procédure utilise ensuite le bloc **répéter** pour ajouter tous les caractères de la chaîne d'entrée (mot) à la chaîne de sortie (motDeSortie), excepté le premier ❷. Ayant sauté le premier caractère, le nombre de répétitions est inférieur à la longueur de notre chaîne d'entrée. À chaque itération de la boucle, un caractère de mot et ajouté à motDeSortie. À la fin de la boucle, la première lettre de mot est ajoutée à motDeSortie ❸, accompagné du suffixe ille ❹.

FAITES LE TEST 8-2

Louchébem.sb2

Ouvrez *Louchébem.sb2* et exécutez-le pour tester cette procédure. L'application demande à l'utilisateur d'entrer un mot et lui donne sa traduction en louchébem. Modifiez-le pour traduire une phrase telle que « Voulez-vous un peu de jus de fruit ? » en louchébem. Indice : faites appel à **Louchébem** à chaque mot pour créer la phrase de sortie. Ensuite, écrivez une procédure qui prend en entrée un mot en louchébem et qui affiche le mot français original.

Corrige mon orthographe

CorrigeMonOrthographe.sb2

Dans cette section, nous allons développer un jeu simple qui génère des mots contenant des fautes d'orthographe et demande au joueur de les corriger. Le jeu créera des mots mal écrits en insérant une lettre aléatoire à une position aléatoire dans un mot français. Bien sûr, il pourrait y avoir plusieurs bonnes réponses. Par exemple, si le mot original est *tour* et que le jeu produit le mot *tjour*, les réponses *jour* et *tour* sont toutes deux correctes. Par simplicité, nous ignorerons cette possibilité et utiliserons le mot original comme bonne réponse.

Tout d'abord, créons une procédure générale pour insérer un caractère à une position spécifique dans une chaîne. Cette procédure, appelée **Insérer**, prend trois paramètres en entrée : la chaîne d'entrée (chaineEntrée), la chaîne (ou le caractère) à insérer (chaineAjout) et la position où vous voulez l'ajouter (posCar). Cette procédure génère une nouvelle chaîne (chaineSortie), comme le montre la figure 8-7.

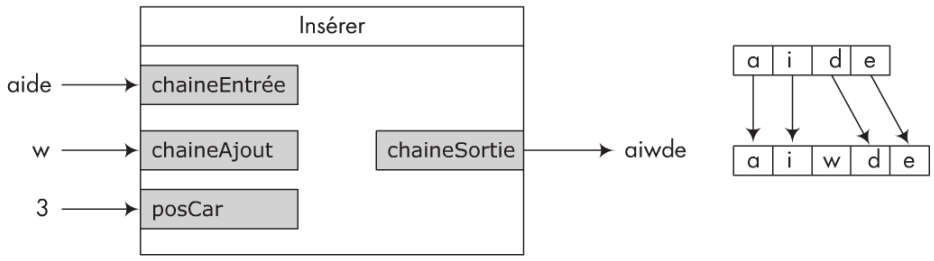


Figure 8-7 : Illustration de la procédure **Insérer**

Copyright © 2016 Eyrolles.

Nous ajouterons les caractères de chaineEntrée un par un à chaineSortie. Lorsque nous atteindrons posCar, nous ajouterons simplement le(s) caractère(s) contenu(s) dans chaineAjout à chaineSortie avant d'y ajouter la lettre qui se situe à posCar dans chaineEntrée. La procédure complète est présentée à la figure 8-8.

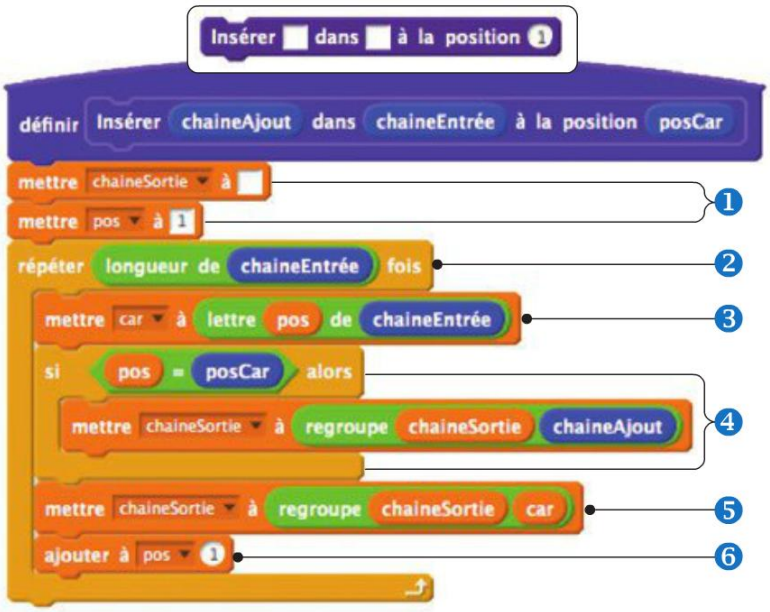


Figure 8-8 : La procédure **Insérer**

Tout d'abord, la procédure initialise chaineSortie en lui affectant une chaîne vide et met pos à 1 pour accéder à la première lettre de chaineEntrée ❶. Elle démarre ensuite une boucle **répéter** pour ajouter une par une les lettres de chaineEntrée à chaineSortie ❷. À chaque itération, la lettre de chaineEntrée pointée par pos est affectée à la variable car ❸. Si la position pointée par pos est égale à celle pointée par posCar, la procédure ajoute chaineAjout à chaineSortie ❹. Dans tous les cas, car est ajoutée à chaineSortie ❺ et pos est incrémentée pour accéder à la lettre suivante de chaineEntrée ❻.

Maintenant que nous avons notre procédure **Insérer**, jetons un œil au script principal du jeu, présenté à la figure 8-9.

Copyright © 2016 Eyrolles.

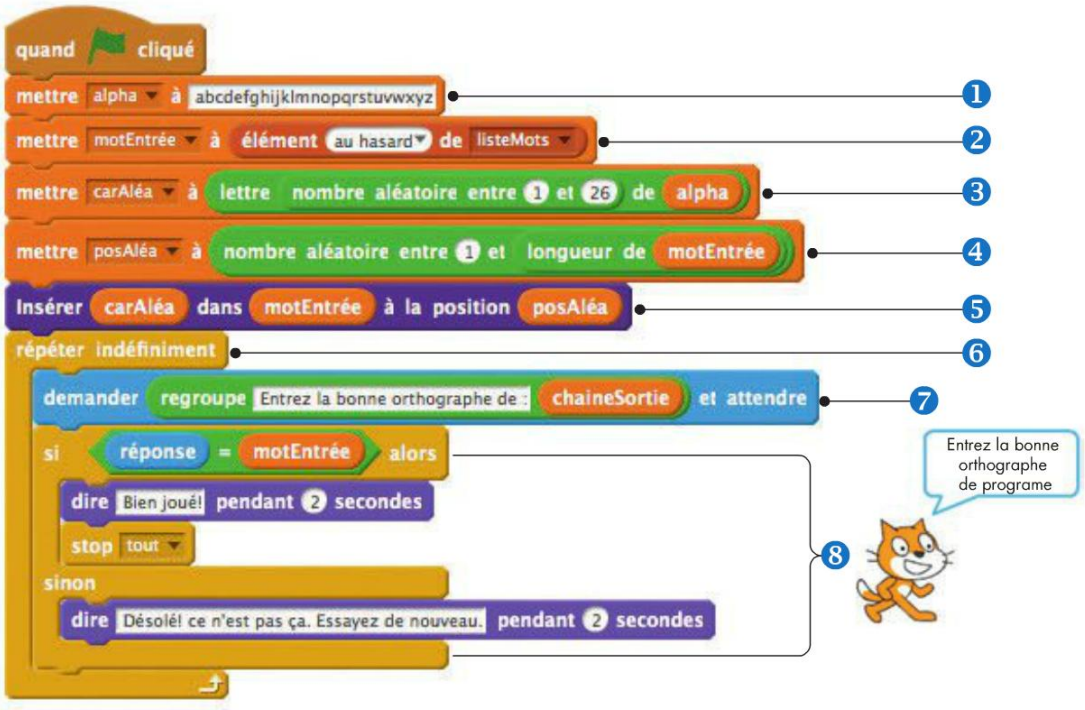


Figure 8-9 : Le script principal du jeu Corrige mon orthographe

La chaîne de caractères alpha contient toutes les lettres de l'alphabet. Elle fournira une lettre aléatoire à ajouter au mot auquel nous désirons ajouter une faute d'orthographe ❶. Le script choisit un mot aléatoirement dans une liste construite préalablement et l'enregistre dans motEntrée ❷. Vous en apprendrez plus sur les listes au chapitre suivant ; pour l'instant, voyez cette liste comme une banque de mots. Le script sélectionne ensuite un caractère aléatoire (carAléa) dans alpha ❸ pour le placer dans motEntrée à une position aléatoire (posAléa) ❹. Il appelle la procédure **Insérer** pour fabriquer le mot mal orthographié (chaineSortie) ❺. Ensuite, il démarre une boucle pour obtenir la réponse de l'utilisateur ❻. Dans cette boucle, le script demande à l'utilisateur d'entrer le mot avec la bonne orthographe ❼ et utilise un bloc **si/sinon** pour vérifier la réponse ❽. Si la réponse correspond au mot original (motEntrée), le jeu se termine ; sinon, le joueur doit réessayer.

CorrigeMonOrthographe.sb2

FAITES LE TEST 8-3

Ouvrez *CorrigeMonOrthographe.sb2* et jouez-y plusieurs fois pour comprendre comment il fonctionne. Pouvez-vous modifier le jeu de façon à ce que le mot corrompu contienne deux lettres additionnelles au lieu d'une ?

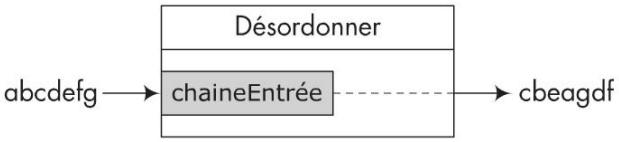
Réordonne

Réordonne.sb2

Notre dernier exemple présente un autre jeu d'orthographe un peu plus compliqué. Nous choisirons un mot français, mélangerons ses lettres et demanderons au joueur de deviner le mot original.

Copyright © 2016 Eyrolles.

Commençons par créer la procédure qui désordonne aléatoirement les caractères d'une chaîne donnée. Le script appelant définit la chaîne d'entrée (chaineEntrée) et la procédure, nommée **Désordonner**, la modifie comme le montre la figure 8-10.



Étape	Nombre aléatoire	Ajouter à chaine1	Supprimer de chaineEntrée
1	3	c	abdefg
2	2	cb	adefg
3	1	cba	defg
4	2	cbae	dfg
...	...	...	...

Figure 8-10 : Illustration de la procédure **Désordonner**

Nous choisirons une lettre aléatoire de chaineEntrée à ajouter à notre *chaîne temporaire*, chaine1, laquelle démarre vide et servira à stocker le mot mélangé lors de sa construction. Nous supprimerons ensuite la lettre de chaineEntrée afin de ne pas la réutiliser et répéterons le processus jusqu'à ce que chaineEntrée soit vide.

La procédure **Désordonner** implémente ces étapes (figure 8-11).

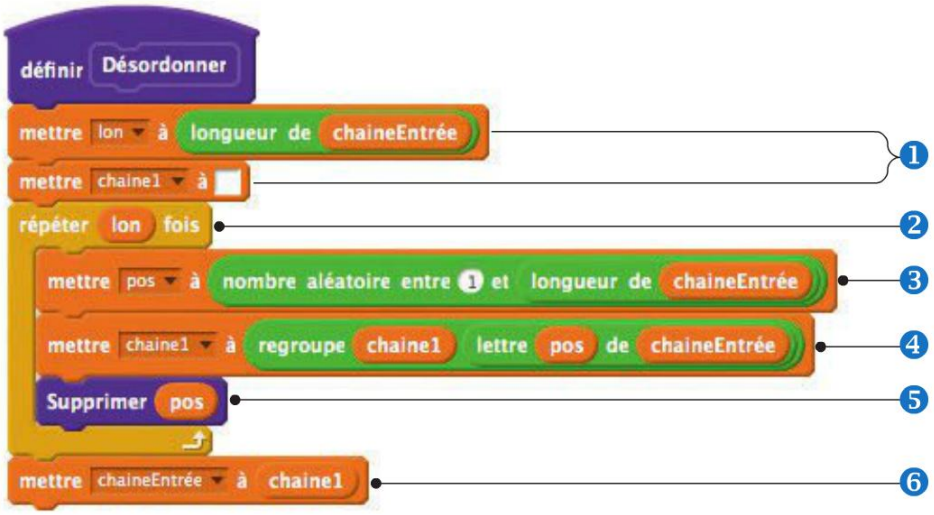


Figure 8-11 : La procédure **Désordonner**

Tout d'abord, **Désordonner** initialise lon à la longueur de la chaineEntrée et vide la chaîne temporaire chaine1 ❶. La procédure démarre ensuite une boucle **répéter** pour mettre en place le mot mélangé ❷. Le nombre de répétitions équivaut à la longueur de la chaîne d'entrée. À chaque itération de la boucle, nous choisissons une position aléatoire dans chaineEntrée ❸ et ajoutons le caractère correspondant à chaine1 ❹. Notez que nous avons utilisé **longueur de** dans l'étape ❸ car chaineEntrée et sa longueur seront modifiés dans la boucle. Ensuite, nous appelons la procédure nommée **Supprimer** pour effacer le caractère que nous venons d'utiliser dans chaineEntrée ❺. Quand la boucle a terminé de recopier les lettres, le mot mélangé (chaine1) est affecté à chaineEntrée ❻.

Copyright © 2016 Eyrolles.

La procédure **Supprimer**, qui évite d'ajouter deux fois la même lettre au mot mélangé, est présentée à la figure 8-12. Elle supprime un caractère de chaineEntrée à la position spécifiée par le paramètre posCar.

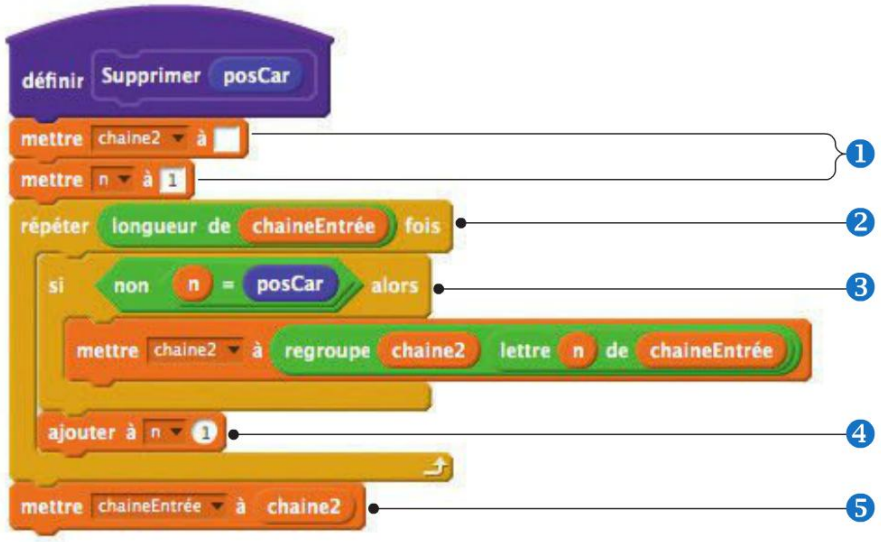


Figure 8-12 : La procédure **Supprimer**

Cette procédure utilise une autre chaîne temporaire, nommée chaine2, pour construire le mot débarrassé de la lettre à supprimer. Elle commence par vider chaine2 et par initialiser à 1 le compteur n de boucle, pour accéder au premier caractère de chaineEntrée ❶. La procédure démarre ensuite une boucle **répéter** pour construire la chaîne de sortie ❷. Si nous ne voulons pas supprimer le caractère actuel, nous l'ajoutons à chaine2 ❸. Le compteur de boucle est ensuite incrémenté pour accéder à la prochaine lettre de chaineEntrée ❹. Lorsque la procédure se termine, chaineEntrée se voit affecter le nouveau mot (chaine2) ❺.

Nous sommes maintenant prêts à étudier le script principal du jeu, présenté à la figure 8-13.

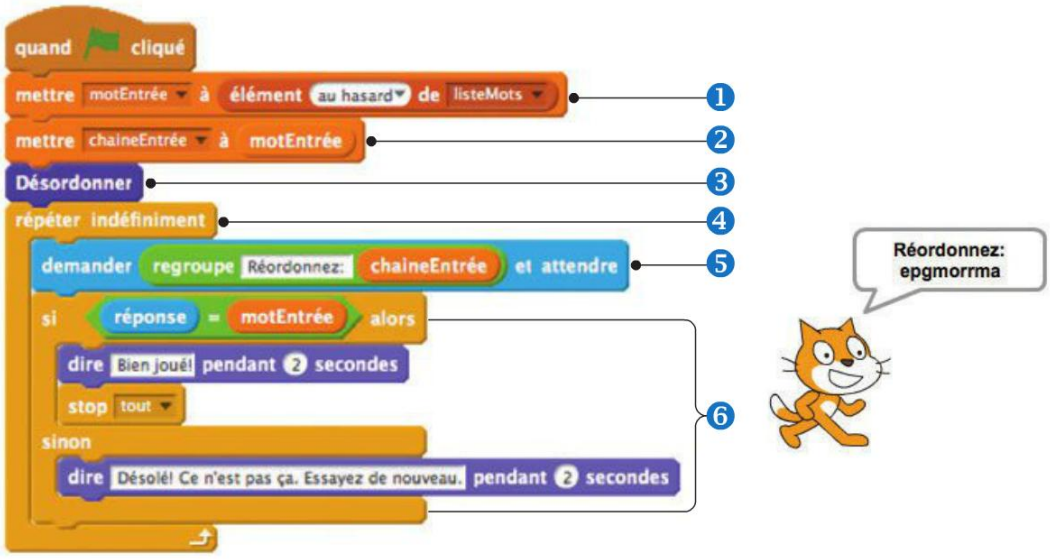


Figure 8-13 : Le script principal du jeu Réordonne

Copyright © 2016 Eyrolles.

Le script sélectionne un mot au hasard dans la liste et le stocke dans motEntrée ❶. Il affecte ensuite motEntrée à chaineEntrée ❷ et appelle **Désordonner** pour changer l'ordre des caractères de cette dernière ❸. Après cela, le script démarre une boucle pour obtenir la réponse du joueur ❹. Dans la boucle, le script demande au joueur de rentrer le mot ordonné ❺ et utilise un bloc **si/sinon** pour vérifier la réponse ❻. Cette partie est identique à celle du jeu Corrige mon orthographe, présenté dans la section précédente.

Les précédents exemples n'ont montré qu'un petit ensemble des différentes opérations que vous pouvez appliquer aux chaînes de caractères. Essayez d'utiliser les mêmes techniques pour modifier des chaînes dans vos propres projets !

Dans le reste de ce chapitre, nous étudierons des programmes qui utilisent des chaînes pour résoudre des problèmes intéressants.

Projets Scratch

Les procédures que vous venez de voir vous ont montré les bases du traitement des chaînes de caractères. Dans cette section, vous appliquerez ce que vous avez appris pour écrire plusieurs applications pratiques. Vous apprendrez alors de nouvelles techniques de programmation que vous pourrez utiliser dans vos propres créations.

Tir

Tir.sb2

Ce jeu est supposé vous apprendre le concept de mouvement relatif de façon stimulante et amusante. Son but est d'estimer l'angle de rotation et la distance à parcourir pour réunir deux objets de la scène. L'interface utilisateur est présentée à la figure 8-14.

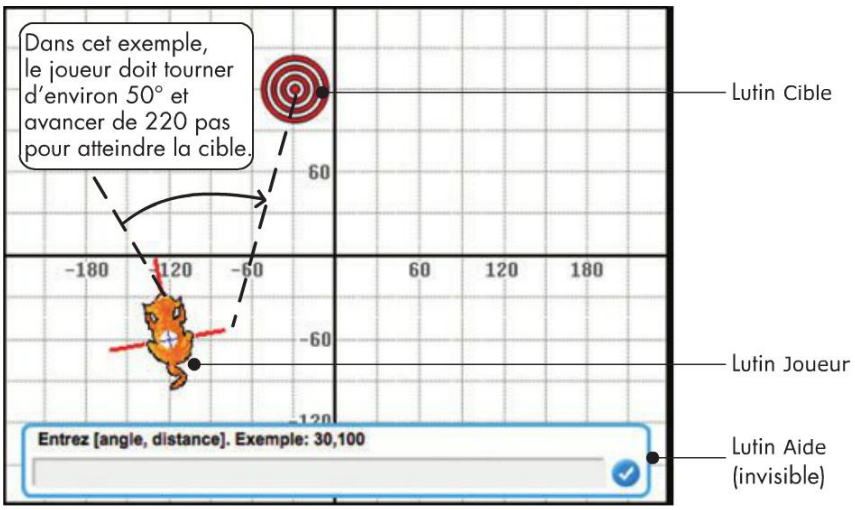


Figure 8-14 : L'interface utilisateur du jeu Tir

Lorsque le jeu démarre, il positionne les lutins Joueur et Cible à des lieux aléatoires de la scène. Il demande ensuite à l'utilisateur d'estimer l'angle de rotation et la distance que Joueur devra parcourir pour atteindre Cible.

Copyright © 2016 Eyrolles.

Le lutin Joueur est alors déplacé selon les nombres entrés par le joueur. Si le lutin s'arrête dans un certain rayon de la cible, le jeu est gagné. Sinon, le lutin Joueur retourne à sa position initiale et le joueur peut faire un nouvel essai. Lorsqu'on clique sur le drapeau vert pour démarrer le jeu, le lutin Joueur exécute le script présenté à la figure 8-15.

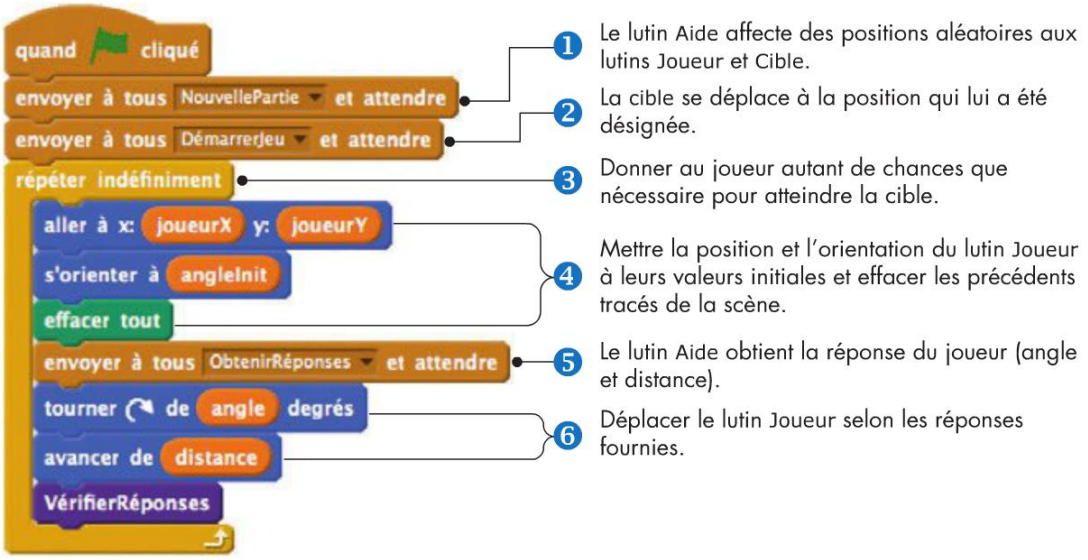


Figure 8-15 : Le script du lutin Joueur *qui s'exécute lorsqu'on clique sur le drapeau vert.*

Le script envoie le message NouveauJeu pour indiquer au lutin Aide d'affecter une nouvelle position aux lutins Joueur et Cible ❶. Le lutin Aide exécute une procédure simple (non présentée) qui met à jour les cinq variables suivantes avec des nombres aléatoires :

- joueurX et joueurY — les coordonnées *x* et *y* du lutin Joueur ;
- cibleX et cibleY — les coordonnées *x* et *y* du lutin Cible ;
- angleInit — l'orientation initiale du lutin Joueur.

Ces variables évitent que Joueur et Cible ne sortent de la scène et font en sorte qu'ils soient espacés d'une certaine distance. Une fois que le script a choisi les nouvelles positions de Joueur et Cible, il envoie le message DémarrerJeu pour déplacer le lutin Cible à sa nouvelle position ❷ (le script de Cible n'est pas présenté ici). Le script entre ensuite dans une boucle infinie donnant au joueur autant de chances que nécessaire pour atteindre la cible ❸. La boucle prend fin grâce à la commande **stop tout** (dans la procédure **VérifierRéponse**) lorsque le joueur a atteint la cible.

À chaque itération, le lutin Joueur est placé à sa position et avec son orientation initiales ; de plus, tout précédent tracé est effacé de la scène ❹. Le script envoie ensuite le message ObtenirRéponses ❺, grâce auquel le lutin Aide demande au joueur de rentrer une réponse (figure 8-16). Le lutin Aide sépare ensuite la réponse en deux parties (avant et après la virgule) et met ainsi à jour l'angle et la distance de Joueur. Suivez les commentaires de la figure 8-16 pour comprendre comment fonctionne ce script.

Copyright © 2016 Eyrolles.

Le lutin Joueur est ensuite déplacé comme le lui a indiqué le joueur, avec son stylo en position d'écriture ❻. Cela laisse une trace du chemin emprunté par le lutin dont le joueur pourra se servir pour affiner son estimation au tour suivant.

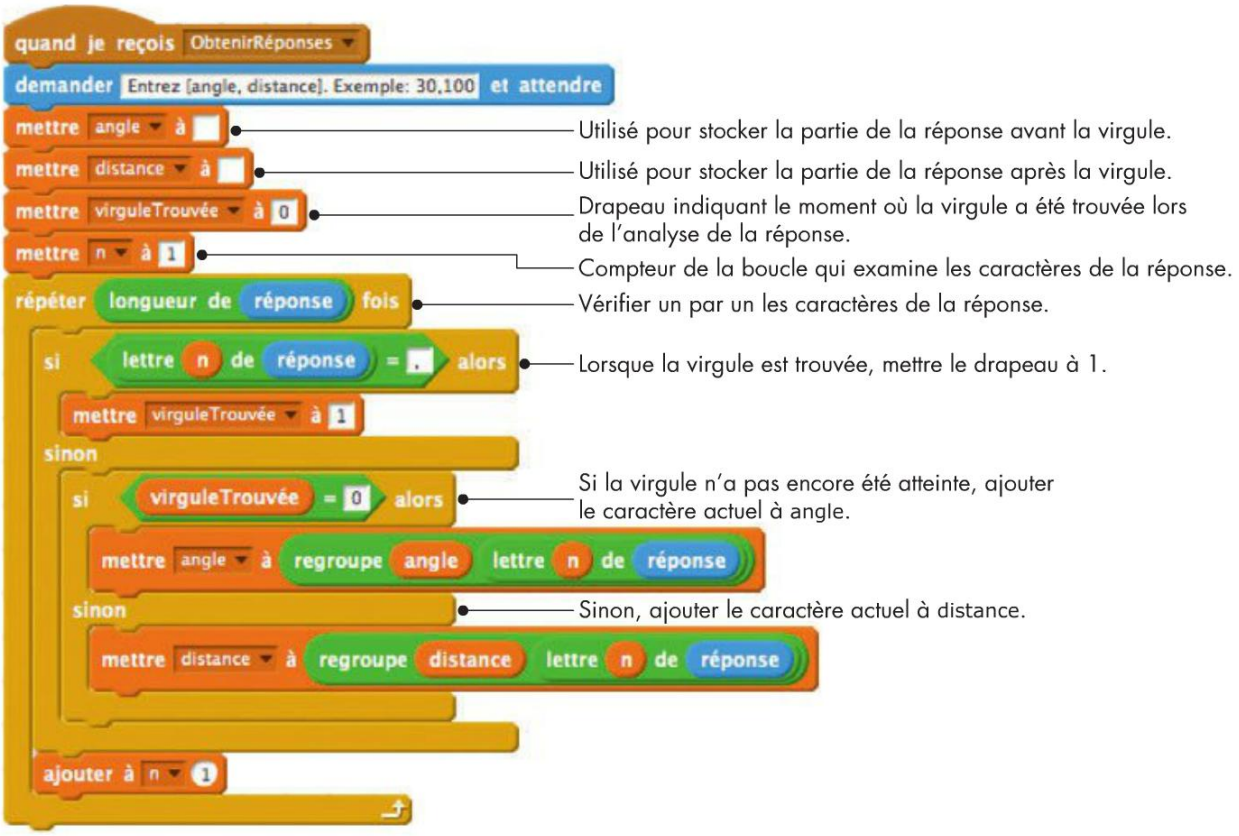


Figure 8-16 : Le script ObtenirRéponses

Enfin, le lutin Joueur exécute la procédure **VérifierRéponses** pour voir s'il est assez près de la cible ou non. Le jeu ne se termine que si le lutin Joueur arrive très près de la cible (figure 8-17).

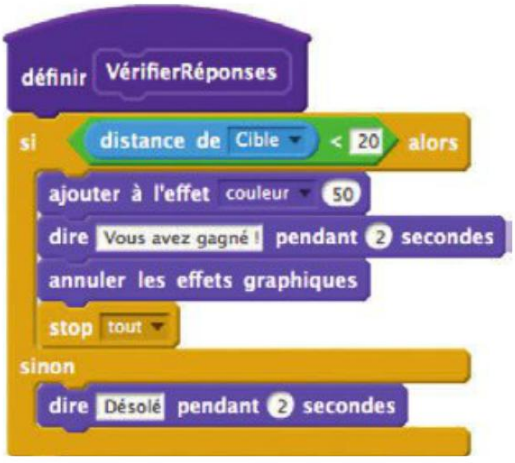


Figure 8-17 : La procédure **VérifierRéponses** *du lutin* Joueur

FAITES LE TEST 8-4

Modifiez le jeu Tir pour affecter un score différent au joueur selon le nombre d'essais qu'il lui aura fallu pour atteindre la cible.

Copyright © 2016 Eyrolles.

Convertisseur binaire-décimal

BinaireDécimal .sb2

Les nombres binaires (à base 2) ne possèdent que deux chiffres : 1 et 0. Les ordinateurs fonctionnent et communiquent avec des nombres binaires. Les humains, quant à eux, préfèrent travailler avec le système décimal (à base 10). Dans cette section, vous allez développer une application qui convertira des nombres binaires en nombres décimaux. Vous pourrez ensuite créer un jeu qui teste votre capacité à faire une telle conversion de tête.

Voyons d'abord comment effectuer une conversion binaire vers décimal. La figure 8-18 montre l'exemple du nombre binaire 10011011.

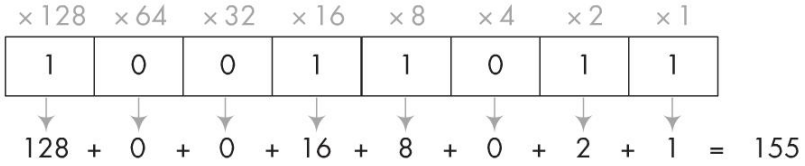


Figure 8-18 : Conversion d'un nombre binaire en un nombre décimal

Nous n'avons qu'à multiplier chaque chiffre binaire par la valeur correspondant à sa position et additionner tous les produits. Les *valeurs positionnelles* correspondent à des puissances de la base croissantes de droite à gauche, la première position ayant la puissance 0. Puisque le système binaire est de base 2, le chiffre le plus à droite a une valeur positionnelle de $2^0 = 1$; vous devrez donc multiplier le chiffre correspondant par 1. Vous devrez multiplier le suivant par $2^1 = 2$, le suivant par $2^2 = 4$ et ainsi de suite.

La figure 8-19 montre l'interface utilisateur de l'application de conversion binaire-décimal. Le programme demande à l'utilisateur de rentrer un nombre binaire de 8 bits. Il affiche ensuite sur la scène le nombre binaire entré avec le lutin Bit, qui utilise deux costumes pour représenter le 1 et le 0. Le programme calcule également le nombre décimal équivalent et le lutin Pilote, qui porte le costume d'ordinateur, affiche cette valeur à l'utilisateur.

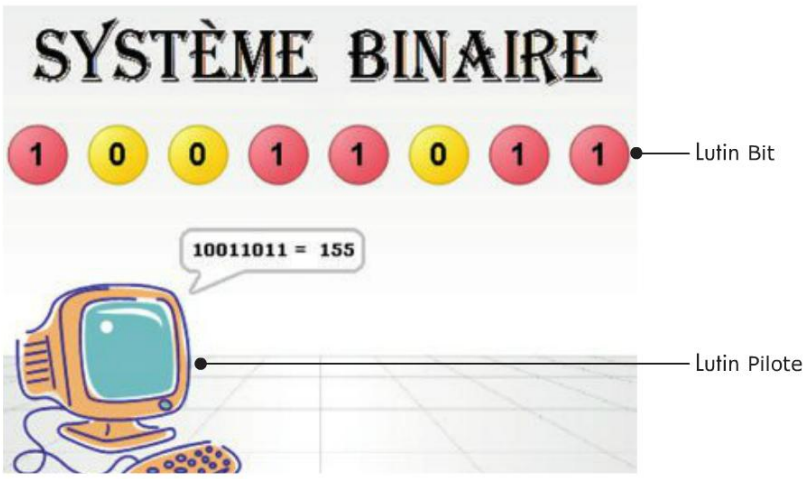


Figure 8-19 : Programme de conversion binaire-décimal

Copyright © 2016 Eyrolles.

FAITES LE TEST 8-5

Effectuez de tête les conversions suivantes, puis vérifiez vos réponses avec l'application : (a) 10101001, (b) 11010010 et (c) 11000011.

Le programme démarre lorsqu'on clique sur le drapeau vert. Cet événement est pris en compte par le lutin Pilote, qui exécute le script de la figure 8-20.

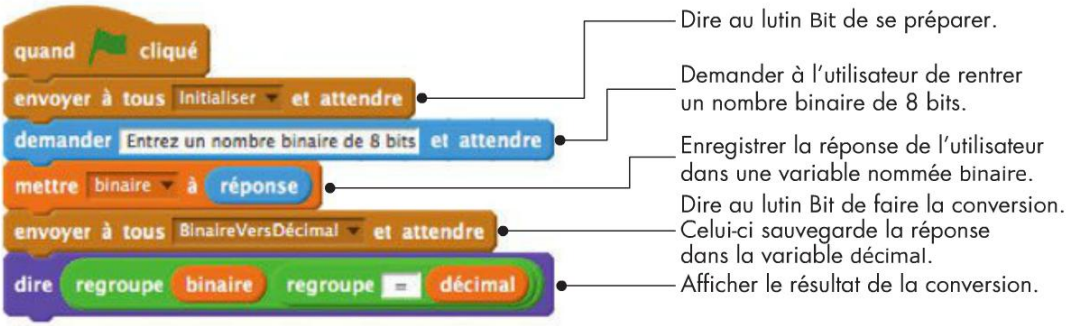


Figure 8-20 : Le script du lutin Pilote

Ce script prépare la scène et demande à l'utilisateur de rentrer un nombre binaire que le lutin Bit devra convertir. Une fois que ce dernier a fini son travail, le lutin Pilote montre à l'utilisateur la valeur décimale, calculée et stockée par le lutin Bit dans une variable partagée nommée décimal.

Le script exécuté par Bit en réponse au message Initialiser est présenté à la figure 8-21.

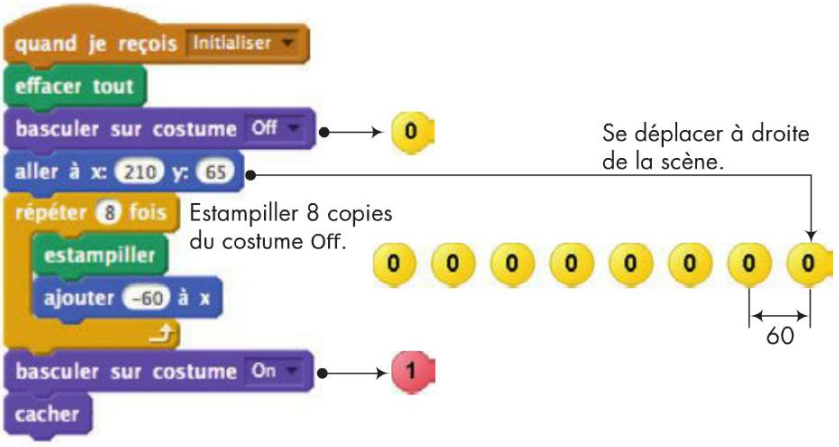


Figure 8-21 : Le script Initialiser *du lutin* Bit

Ce script dessine un motif de bits représentant huit zéros sur la scène. Comme vous le verrez dans un instant, lorsque l'utilisateur place un 1 dans le chiffre binaire d'entrée, le script estampillera un costume du chiffre 1

Copyright © 2016 Eyrolles.

au dessus du bit correspondant. Lorsque l'utilisateur entre le nombre binaire à convertir, le lutin Bit reçoit le message BinaireVersDécimal et exécute le script de la figure 8-22.

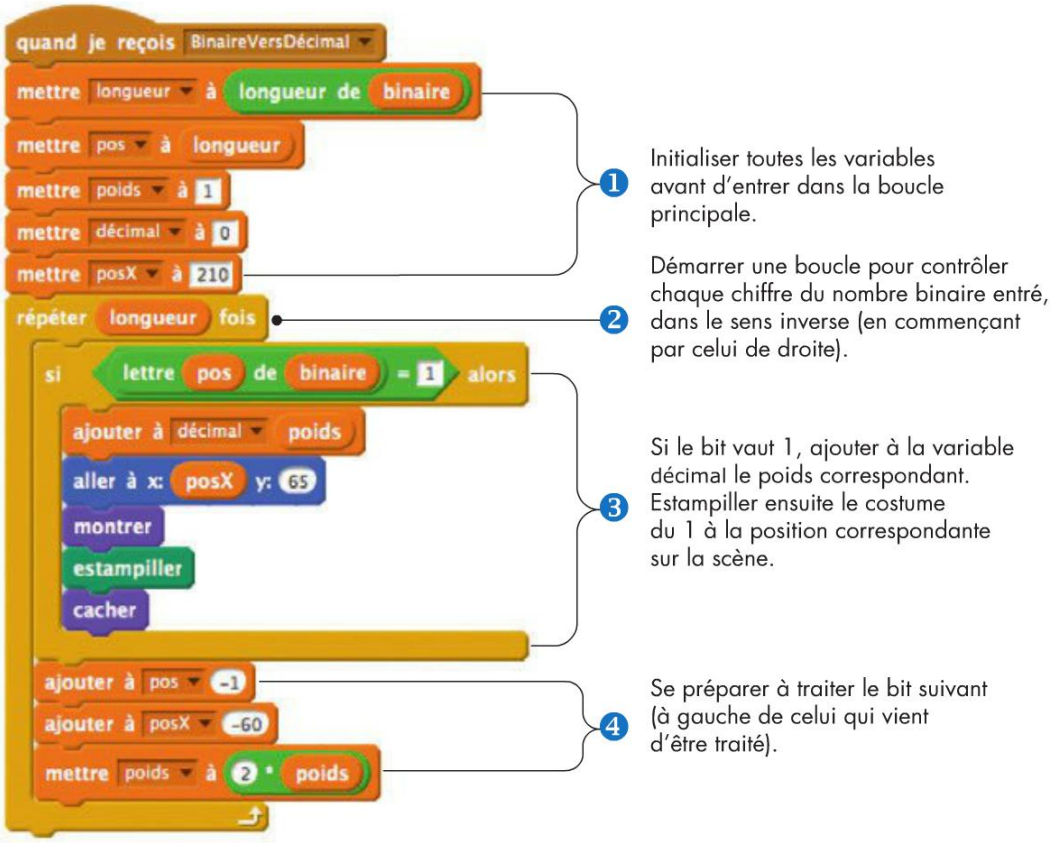


Figure 8-22 : Le script BinaireVersDécimal *du lutin* Bit

Tout d'abord, la procédure de conversion initialise toutes les variables qu'elle utilisera ❶ :

- longueur est le nombre de bits contenus dans le nombre binaire entré par l'utilisateur.
- pos pointe le bit le plus à droite du nombre binaire entré.
- poids est mis à la valeur positionnelle du bit le plus à droite.
- décimal est mis à 0 et contiendra à la fin le résultat de la conversion.
- posX est initialisé à la position *x* du bit le plus à droite.

À l'intérieur de la boucle **répéter** ❷, la procédure contrôle chaque bit pour vérifier si c'est un 1 ou un 0. Si la boucle trouve un 1 ❸, elle ajoute la valeur actuelle de poids à décimal et estampille le costume bit-1 au dessus du costume bit-0.

À la fin de la boucle, le script met à jour plusieurs variables avant de passer à l'itération suivante :

Copyright © 2016 Eyrolles.

- pos est mis à jour pour pointer sur le bit à gauche de celui qui vient d'être traité.
- posX est aligné au centre de l'image du bit suivant, au cas où nous voudrions estampiller une nouvelle image.
- poids est multiplié par 2, ce qui signifie qu'il prendra les valeurs 1, 2, 4, 8, 16 et ainsi de suite jusqu'à la fin de la boucle.

FAITES LE TEST 8-6

Faites en sorte que le lutin Pilote valide le nombre entré par l'utilisateur avant d'envoyer le message BinaireVersDécimal au lutin Bit. Vous devrez vérifier que (1) le nombre entré par l'utilisateur est bien binaire (donc qu'il ne contient que des uns et des zéros) et (2) il n'est pas plus long que huit bits.

Le jeu du Pendu

Pendu.sb2

Dans cette section, nous allons développer le fameux jeu du Pendu (figure 8-23).

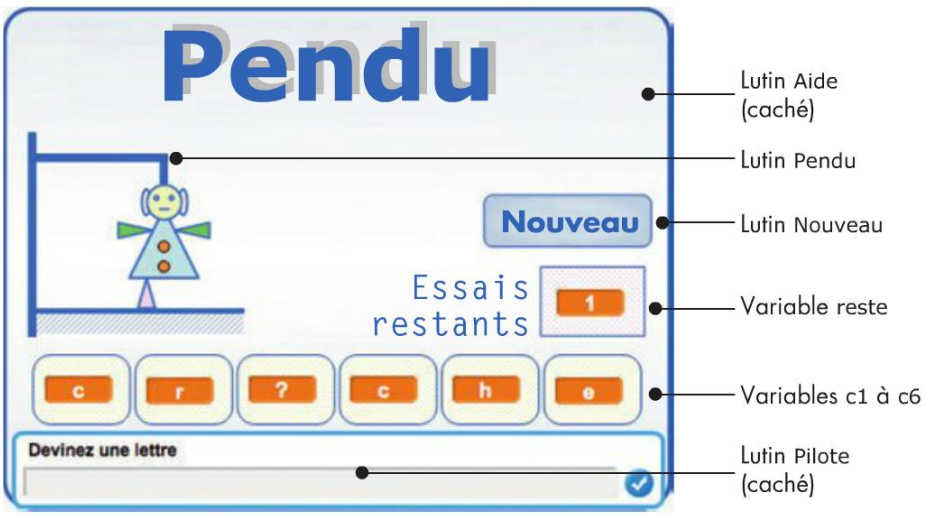


Figure 8-23 : L'interface utilisateur du jeu du Pendu

Le programme sélectionne aléatoirement un mot secret de six lettres et affiche un point d'interrogation pour chacune d'entre elles. Le joueur dispose de huit chances pour trouver toutes les lettres du mot. Si le joueur trouve une lettre contenue dans le mot secret, le programme affiche tous les emplacements où se trouve cette lettre. Sinon, il affiche une nouvelle partie du l'image du pendu (tête, corps, bras gauche, bras droit, etc.). Après huit réponses fausses, le pendu est entièrement dessiné et le joueur a perdu. Si, au contraire, il trouve toutes les lettres en huit essais ou moins, alors il a gagné. Cette application possède les lutins suivants :

Copyright © 2016 Eyrolles.

- Pilote. Ce lutin se cache lorsque le jeu démarre, demande au joueur de rentrer une lettre et traite sa réponse. Lorsque le jeu prend fin, il affiche l'un des deux costumes suivants :



- Pendu. Ce lutin affiche l'évolution de l'image du pendu. Il possède un total de neuf costumes, chacun d'entre eux possédant une partie supplémentaire du corps du pendu, comme le montre la figure 8-24.

- Nouveau. Ce lutin affiche le bouton Nouveau sur la scène.

- Aide. Ce lutin invisible affiche les lettres trouvées par le joueur, ainsi que le nombre d'essais restants. Il utilise sept variables dont les moniteurs sont configurés en mode grande lecture et positionnés à l'emplacement adéquat sur la scène. Utiliser un lutin différent pour mettre à jour les affichages sépare la logique du jeu de l'interface utilisateur. Vous pouvez, par exemple modifier le lutin pour afficher des lettres plus fantaisistes sur la scène sans modifier le reste de l'application.

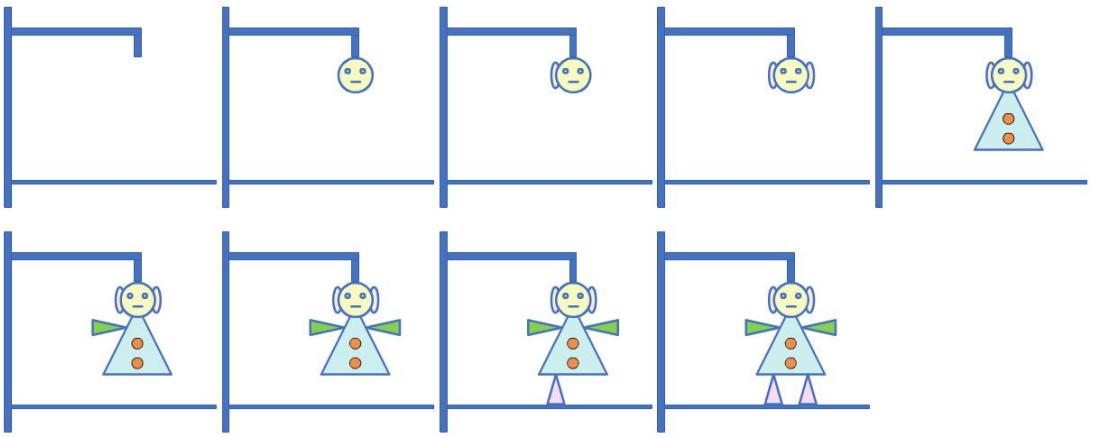


Figure 8-24 : Les neuf costumes du lutin Pendu

Lorsque le joueur appuie sur le bouton Nouveau, le lutin correspondant envoie un message NouvellePartie pour alerter le lutin Pilote, qui exécute alors le script de la figure 8-25.

Copyright © 2016 Eyrolles.



Figure 8-25 : Le script de gestion du message NouvellePartie *du lutin* Pilote

Le script réinitialise l'interface utilisateur ❶ et démarre une boucle ❷ pour lire les réponses du joueur. Une autre procédure appelée par le lutin Pilote mettra fin à la boucle grâce au bloc **stop tout** lorsque la condition de fin de jeu est détectée.

À chaque itération de la boucle, le lutin Pilote demande au joueur de deviner une lettre et attend sa réponse ❸. Lorsque cette dernière est entrée, le script appelle **TraiterRéponse**, qui met à jour le drapeau nommé lettrePrésente pour indiquer si la lettre entrée est contenue dans le mot ou non.

Lorsque **TraiterRéponse** prend fin, le script vérifie le drapeau lettrePrésente ❹ et agit en fonction de la réponse du joueur. Je vais à présent expliquer les procédures appelées par NouvellePartie, en commençant par les scripts de la figure 8-26.

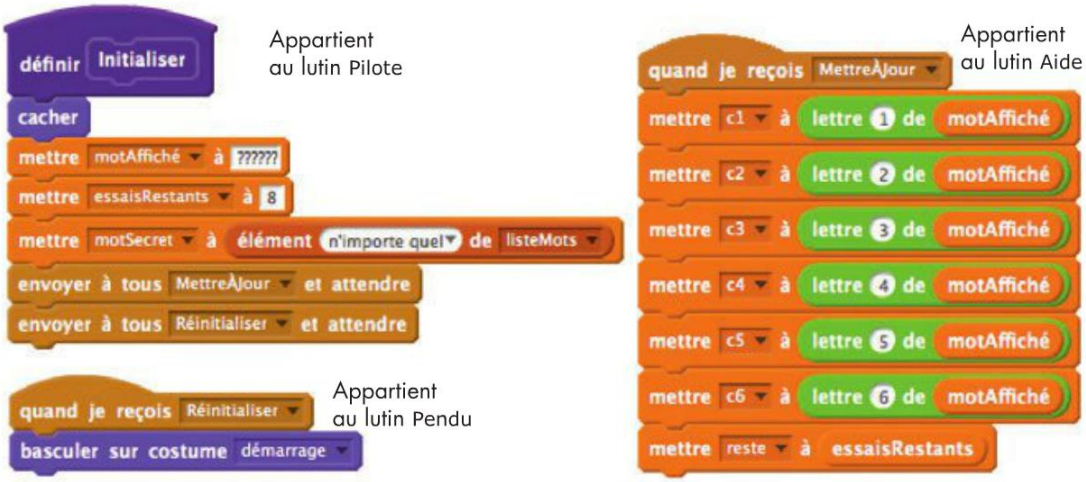


Figure 8-26 : Les scripts déclenchés par la procédure **Initialiser**

Pendant l'initialisation, le lutin Pilote se cache, initialise motAffiché en lui affectant une chaîne de caractères contenant six points d'interrogation et met essaisRestants à 8. Il sélectionne ensuite le motSecret à partir d'une liste prédéfinie de mots de six lettres. La procédure envoie ensuite le message MettreÀJour pour indiquer au lutin Aide d'affecter les nouvelles valeurs

Copyright © 2016 Eyrolles.

à ses variables (qui sont affichées sur la scène). La dernière instruction envoie le message Réinitialiser au lutin Pendu, qui bascule vers son costume démarrage (une potence vide).

Considérons à présent un exemple simple qui nous permettra de comprendre ce que fait la procédure TraiterRéponse (figure 8-27). Supposez que le mot secret soit *cruche* et que ce soit le premier tour du jeu (ce qui signifie que motAffiché est « ?????? »). Si la première réponse du joueur est r, **TraiterRéponse** change lettrePrésente à 1 pour indiquer que la réponse est correcte, motAffiché à « ?r???? » pour montrer l'emplacement de la lettre et nbPoindint (le nombre de points d'interrogation contenus dans la nouvelle chaîne de caractères affichée) à 5. Lorsque nbPoindint atteint 0, le joueur a trouvé toutes les lettres de mot secret. **TraiterRéponse** appartient au lutin Pilote et vous pouvez voir son script complet sur la figure 8-27 (à gauche).

TraiterRéponse commence par remettre nbPointdint et le drapeau lettrePrésente à 0. Cette procédure incrémente nbPointdint de 1 à chaque lettre inconnue dans le mot secret. La variable temporaire temp, utilisée pour construire la chaîne de caractères à afficher après chaque réponse, se voit affecter une chaîne vide. La variable pos est utilisée comme compteur de boucle.

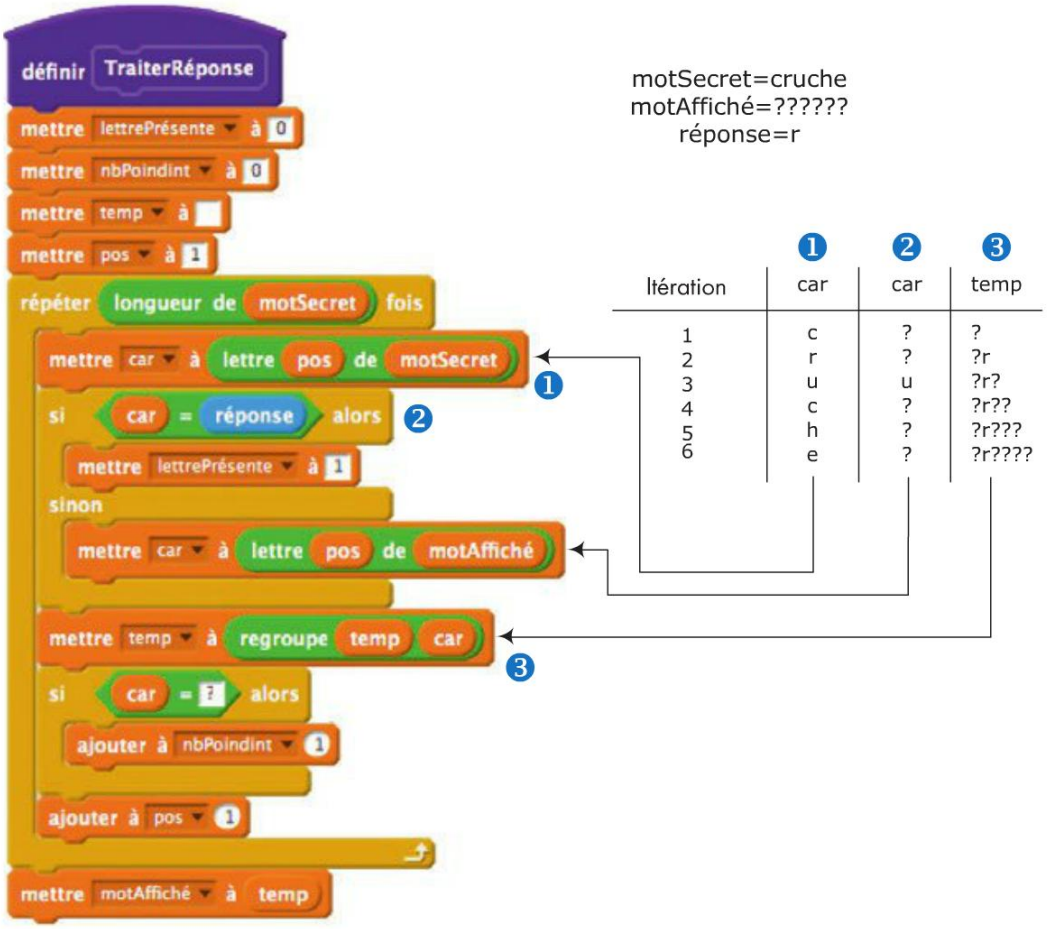


Figure 8-27 : La procédure **TraiterRéponse**

La boucle examine chaque lettre de motSecret, en utilisant pos comme indice. Si la lettre examinée (contenue dans car) est égale à la lettre entrée par l'utilisateur (contenue dans la variable intégrée de Scratch réponse), le

Copyright © 2016 Eyrolles.

drapeau lettrePrésente est mis à 1. Sinon, la variable car est affectée à la lettre se trouvant à l'emplacement correspondant de la variable motAffiché. Dans tous les cas, le script ajoute car à la fin de temp, comme le montre la figure 8-27 (à droite).

Lorsque la boucle se termine, la variable motAffiché contient les six lettres à afficher sur la scène, en prenant en compte la dernière réponse du joueur. La boucle vérifie également le nombre de points d'interrogation restants dans la chaîne de caractères. S'il n'y en a plus, alors l'utilisateur a bien deviné le mot secret.

Lorsque **TraiterRéponse** prend fin, le script de gestion du message NouvellePartie vérifie le drapeau lettrePrésente pour voir si le joueur a rentré une bonne réponse. Si ce n'est pas le cas, la procédure **TraiterMauvaiseRéponse** (figure 8-28) est appelée.

Cette procédure envoie le message MauvaiseRéponse pour indiquer au lutin Pendu de basculer vers le costume suivant, puis de décrémenter le nombre d'essais restants de 1. Si l'utilisateur a épuisé tous ses essais, le script lui révèle le mot secret et met fin au jeu. Sinon, il envoie le message MettreÀJour pour montrer à l'utilisateur combien il lui reste d'essais.

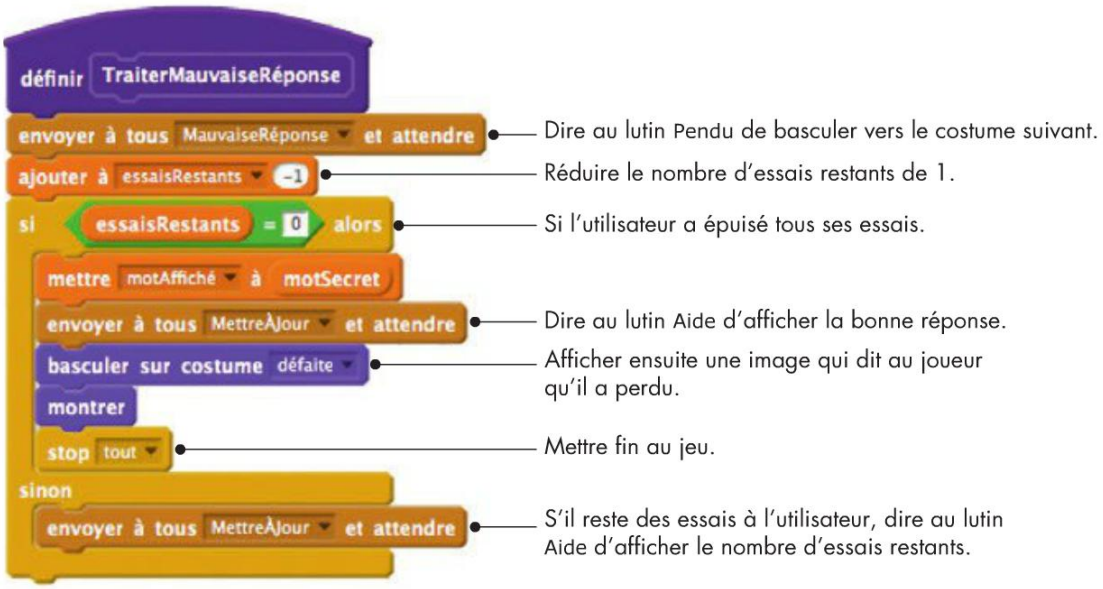


Figure 8-28 : La procédure **TraiterMauvaiseRéponse**

Si la lettre entrée par le joueur est correcte, la procédure **TraiterBonneRéponse** (figure 8-29) est appelée au lieu de **TraiterMauvaiseRéponse**.

Copyright © 2016 Eyrolles.

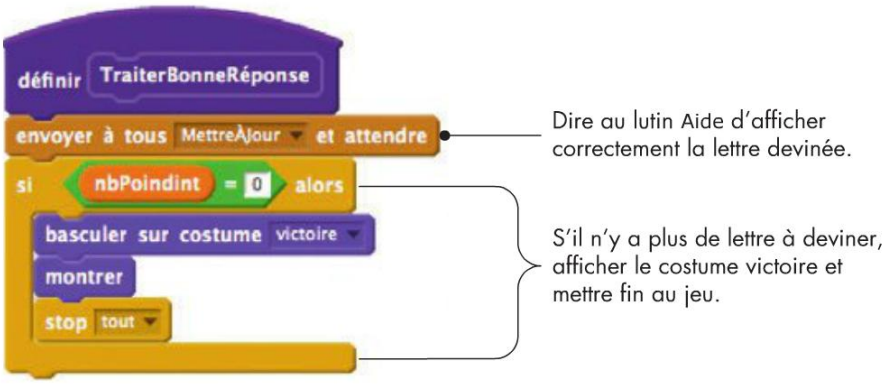


Figure 8-29 : La procédure **TraiterBonneRéponse**

La procédure **TraiterBonneRéponse** envoie le message MettreÀJour pour afficher la lettre que le joueur a devinée correctement. Elle vérifie ensuite la valeur de nbPointdint ; si elle est nulle, le joueur a trouvé toutes les lettres du mot secret et le lutin Pilote affiche le costume victoire avant de mettre fin au jeu.

FAITES LE TEST 8-7

Le programme du jeu du pendu ne valide pas l'entrée utilisateur ; vous pourriez rentrer un caractère non alphabétique ou même un mot entier. Modifiez le programme afin qu'il rejette toute entrée invalide.

Professeur de fractions

ProfesseurDeFractions .sb2

Nous présenterons comme dernier exemple un jeu éducatif permettant d'enseigner les fractions (figure 8-30). Le joueur sélectionne une opération (+, –, × ou ÷) et clique sur le bouton Nouveau pour créer un nouveau problème. Lorsque le joueur entre une réponse et clique sur le bouton Vérifier, le lutin Enseignante (image d'une femme) vérifie la réponse et fournit un message de retour approprié.

Copyright © 2016 Eyrolles.

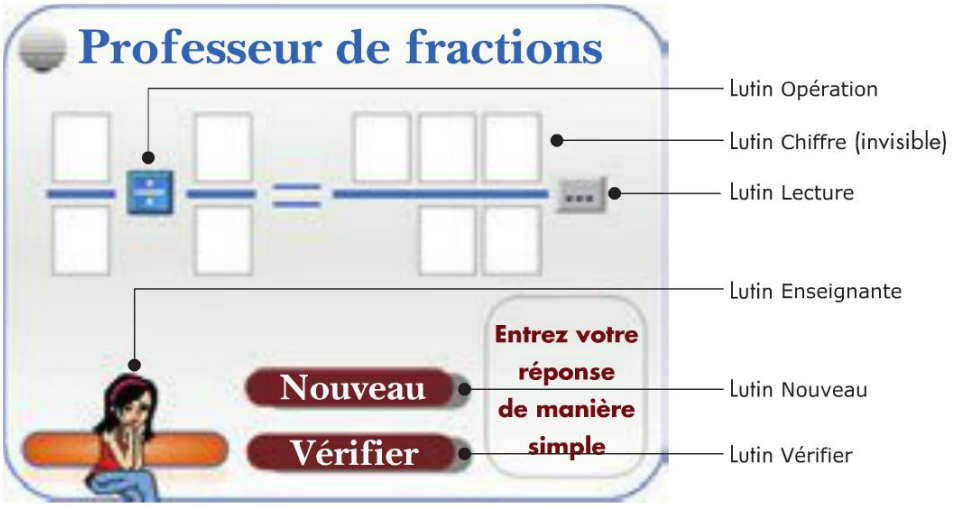


Figure 8-30 : L'interface utilisateur de l'application Professeur de fractions

L'application contient six lutins. Opération laisse le joueur choisir l'opération mathématique. Lecture affiche le bouton permettant d'entrer une réponse. Nouveau et Vérifier affichent les boutons de même nom. Le lutin Enseignante vérifie la réponse du joueur et un lutin invisible nommé Chiffre estampille sur la scène les nombres qui correspondent au problème actuel.

Lorsque le joueur clique sur le bouton Nouveau, le lutin correspondant exécute le script de la figure 8-31. Ce dernier affecte des valeurs aléatoires comprises entre 1 et 9 au numérateur et au dénominateur de chaque opérande, qui sont représentés par les variables num1, den1, num2 et den2. Il envoie ensuite le message NouveauProblème pour indiquer au lutin Chiffre d'estampiller ces nombres sur la scène.



Figure 8-31 : Le script du lutin Nouveau

Le lutin Chiffre possède 12 costumes (nommés d1 à d12), comme le montre la figure 8-32 (à droite). Lorsqu'il reçoit le message NouveauProblème, il estampille les costumes représentant les numérateurs et dénominateurs des deux opérandes. La figure 8-32 montre également la procédure effectuant l'estampillage.

Copyright © 2016 Eyrolles.

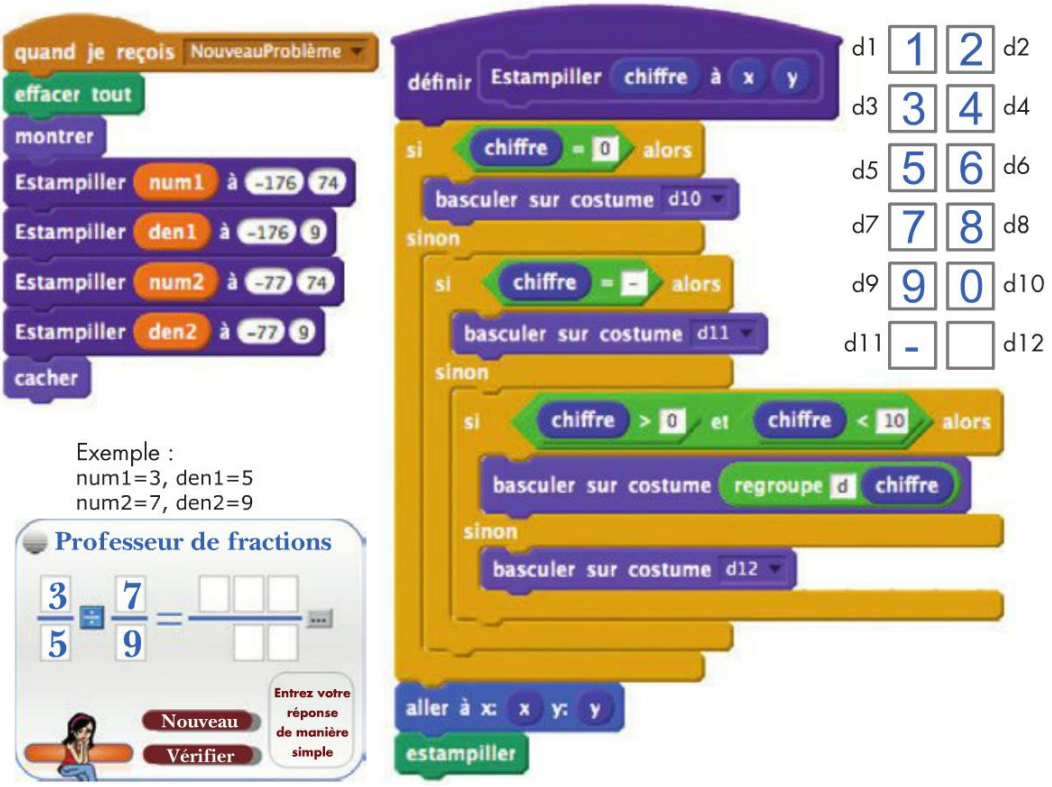


Figure 8-32 : La fonction du lutin Chiffre

La procédure utilise des blocs **si/sinon** imbriqués pour déterminer quel costume correspond au chiffre à estampiller. Notez la façon dont le nom de costume des chiffres 1 à 9 est formé grâce à l'opérateur **regroupe**. Après avoir basculé vers le costume adéquat, le lutin Chiffre se déplace à la position (*x*,*y*) spécifiée et estampille l'image de son costume à cet endroit.

Lorsque le nouveau problème est affiché, l'utilisateur peut cliquer sur le bouton Lecture afin d'entrer une réponse. Le script associé à ce bouton est présenté à la figure 8-33. La partie du script qui sépare la réponse du joueur en deux entités (numérateur et dénominateur) est similaire à celle de la figure 8-16 qui extrait angle et distance de la réponse dans le jeu de Tir ; elle ne sera donc pas présentée ici. Regardez le fichier *ProfesseurDeFractions.sb2* pour voir la procédure complète.

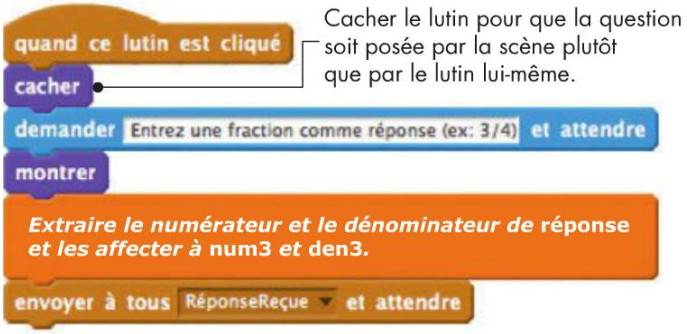


Figure 8-33 : Le script du lutin Lecture

Copyright © 2016 Eyrolles.

D'abord, le script demande à l'utilisateur d'entrer une réponse sous forme d'une fraction (comme, 3/5 ou -7/8). Le script extrait ensuite le numérateur et le dénominateur de la chaîne de caractères entrée (séparés par le signe de division) et les affecte aux variables num3 et den3, respectivement. Par exemple, si l'utilisateur entre -23/15, num3 vaudra -23 et den3 sera égal à 15. Puis le script envoie le message RéponseReçue pour indiquer au lutin Chiffre d'afficher sur la scène la réponse de l'utilisateur. Lorsque Chiffre reçoit ce message, il estampille les chiffres de num3 et den3 aux emplacements correspondants sur la scène, tout comme pour le dénominateur et le numérateur des deux opérandes. Regardez le fichier *ProfesseurDeFraction.sb2* pour les détails.

Après avoir entré la réponse, l'utilisateur peut cliquer sur le bouton Vérifier pour voir si sa réponse est correcte. Le script du lutin Vérifier envoie le message VérifierRéponse pour informer les autres lutins de la demande de l'utilisateur. Ce message est traité par le lutin Enseignante, qui exécute le script de la figure 8-34.

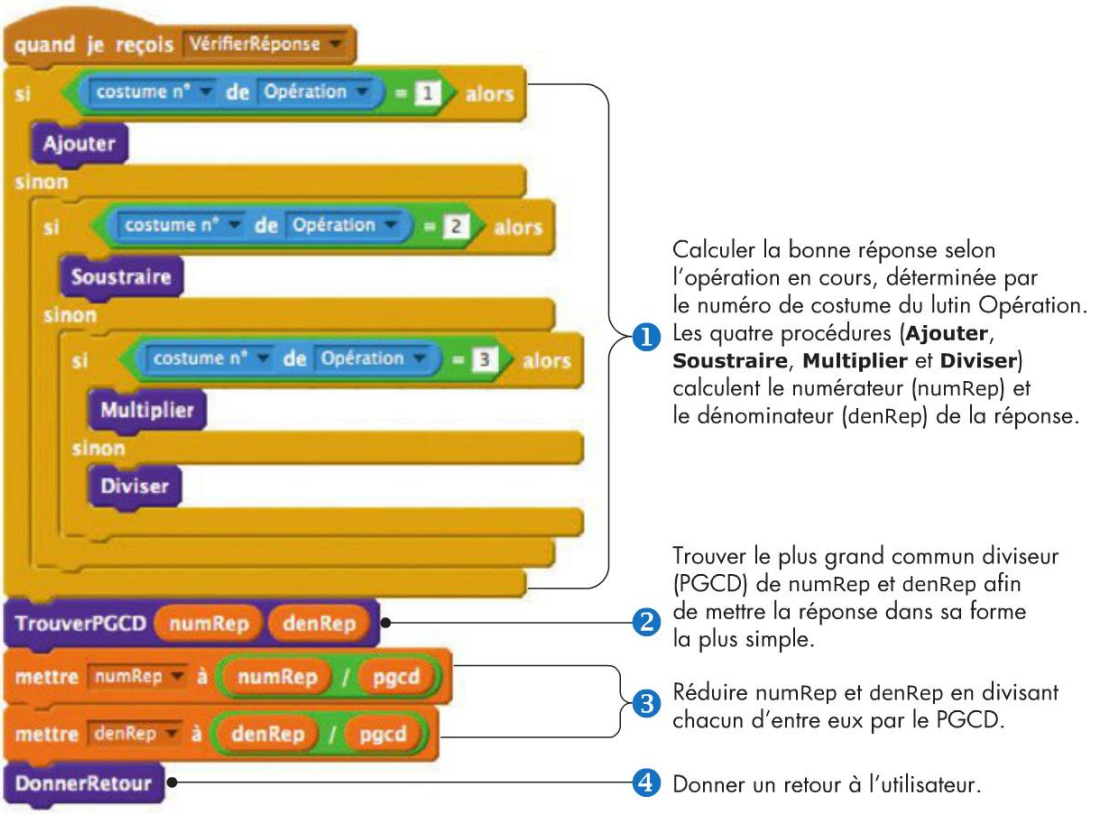


Figure 8-34 : Le script VérifierRéponse

Le costume actuel du lutin Opération indique quelle procédure d'opération exécuter (**Ajouter**, **Soustraire**, **Multiplier** ou **Diviser**) ❶. Les opérations prennent num1, den1, num2 et den2 en entrée et règlent les valeurs de numRep et denRep, qui représentent le numérateur et le dénominateur de la bonne réponse, respectivement. Les quatre procédures sont présentées à la figure 8-35.

Après avoir trouvé le résultat, VérifierRéponse doit l'écrire sous sa forme la plus simple. Par exemple, 2/4 devrait être simplifié à 1/2. Pour cela, le script trouve d'abord le plus grand commun diviseur (PGCD) du numérateur et du dénominateur ❷. Nous verrons ces procédures dans un instant.

Copyright © 2016 Eyrolles.

Après avoir trouvé le PGCD, le script divise numRep et denRep par cette valeur ❸ et appelle **DonnerRetour** ❹ pour indiquer au joueur s'il a rentré la bonne réponse ou non.

À présent, regardons de plus près les détails de ces procédures, en commençant par les quatre opérations de la figure 8-35. Ces procédures calculent le résultat d'une opération de la forme suivante :

$$\frac{num1}{den1}\left[+,-,\times,\div\right]\frac{num2}{den2}=\frac{ansNum}{ansDen}$$

Elles stockent le résultat dans deux variables (numRep et denRep) correspondant au numérateur et au dénominateur de la réponse, respectivement.

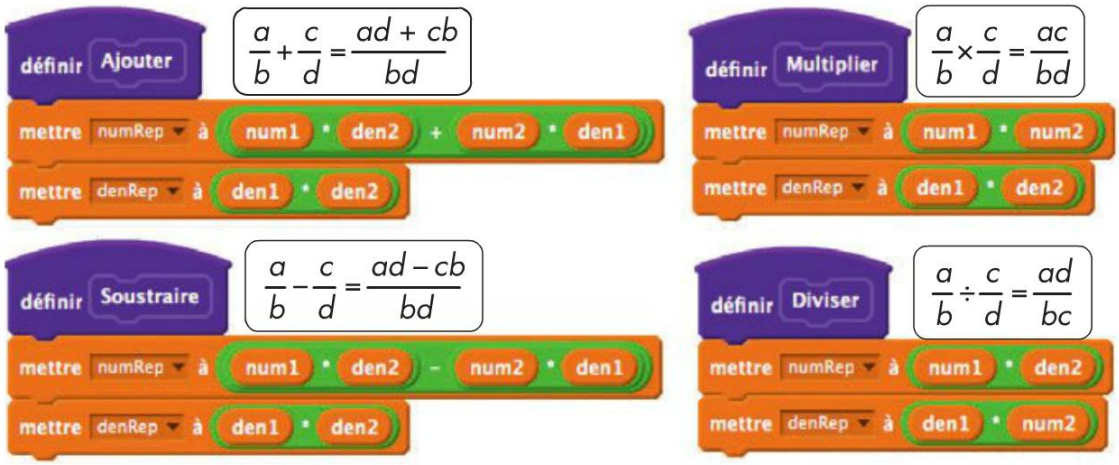


Figure 8-35 : Les procédures **Ajouter**, **Soustraire**, **Multiplier** *et* **Diviser** *du lutin* Enseignante

Passons maintenant à la procédure **TrouverPGCD** (figure 8-36).

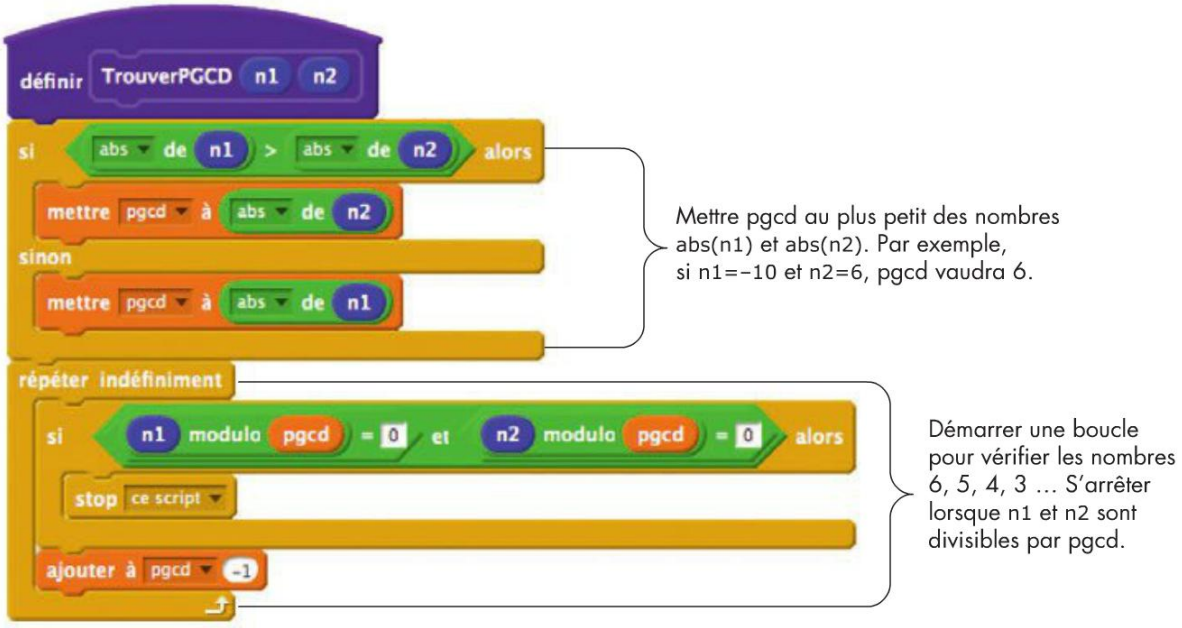
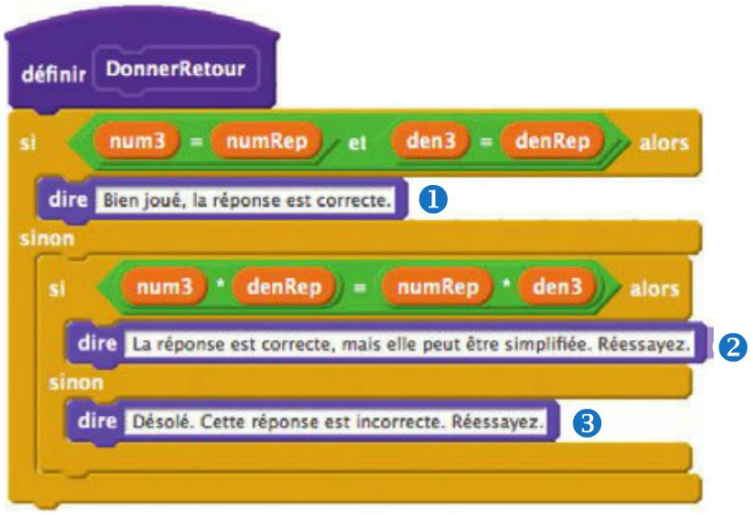


Figure 8-36 : La procédure **TrouverPGCD** *du lutin* Enseignante

Copyright © 2016 Eyrolles.

Suivons le cheminement de la procédure **TouverPGCD** lorsque `n1=-10` et `n2=6`. Nous devons trouver le plus grand entier positif qui divise `n1` et `n2` avec un reste nul. La procédure démarre en mettant `pgcd`, le résultat, à la plus petite valeur absolue des deux nombres traités, 6 dans notre exemple. Une boucle teste ensuite les nombres 6, 5, 4 et ainsi de suite, jusqu'à ce que `n1` et `n2` soient tous deux divisibles par le nombre testé, qui correspondra alors au résultat que nous cherchons. Dans notre exemple, `pgcd` vaudra 2 car les deux nombres (-10 et 6) sont divisibles par 2.

La dernière procédure à étudier est **DonnerRetour**, qui compare la réponse entrée par l'utilisateur avec la bonne réponse et qui affiche un message approprié (figure 8-37). La figure présente également des exemples qui démontrent les différents cas de la structure **si/sinon**.



	Exemples	
	Bonne Réponse $\frac{numRep}{denRep}$	Réponse Utilisateur $\frac{num3}{den3}$
1	$\frac{3}{4}$	$\frac{3}{4}$
2	$\frac{3}{4}$	$\frac{6}{8}$
3	$\frac{3}{4}$	$\frac{2}{3}$

Figure 8-37 : La procédure **DonnerRetour** *du lutin* Enseignante

FAITES LE TEST 8-8

Modifiez le programme d'enseignement de fraction afin de garder une trace du nombre de bonnes et de mauvaises réponses. Faites en sorte de calculer le score et de le montrer à l'utilisateur.

Résumé

Le traitement de chaînes de caractères est une compétence de programmation importante. Dans ce chapitre, vous avez appris à accéder à des caractères individuels d'une chaîne pour les combiner, les comparer, les supprimer et les mélanger.

Nous avons d'abord regardé en détail le type de données « chaînes de caractères » ainsi que la façon dont ces chaînes sont stockées, sous la forme de séquences de caractères. Nous avons ensuite écrit plusieurs procédures illustrant des techniques de base de manipulation de chaînes. Après cela,

Copyright © 2016 Eyrolles.

nous nous sommes servis de ces techniques pour écrire plusieurs applications pratiques et intéressantes. Les concepts développés dans ces projets peuvent être appliqués à beaucoup d'autres domaines et j'espère bien qu'ils vous apporteront des idées pour vos futurs projets.

Dans le chapitre suivant, vous étudierez les listes et la façon de les utiliser pour stocker et manipuler des groupes de données. Munis de cette nouvelle structure de données, vous posséderez tous les outils nécessaires à la création de programmes professionnels dans Scratch.

Problèmes

1. Écrivez un programme qui demande à l'utilisateur d'entrer un mot puis qui répète *N* fois ce mot, avec *N* le nombre de lettres de la réponse.
2. Écrivez un programme qui demande à l'utilisateur de rentrer un mot. Le programme détermine ensuite le nombre d'apparitions de la lettre *a* dans le mot entré.
3. Écrivez un programme qui demande à l'utilisateur de rentrer un nom singulier en français. Le programme produira ensuite la forme plurielle de ce nom. Indice : vérifiez la dernière et l'avant-dernière lettres du mot entré. Pour que le programme reste simple, ne considérez que les règles suivantes : ajouter un *s* à la fin du mot, sauf si le mot finit par *al* (vous remplacerez alors *al* par *aux*), *au* ou *eu* (auquel cas vous ajouterez un *x*).
4. Écrivez un programme qui demande à l'utilisateur d'entrer un caractère unique (entre *a* et *z*) et qui lui indique sa position dans l'alphabet (*a* = 1, *b* = 2, *c* = 3, etc.). Les lettres majuscules et minuscules doivent être traitées de la même façon. Indice : définissez une variable nommée alpha qui contiendra les lettres de l'alphabet, comme nous l'avons fait sur la figure 8-9, puis utilisez une boucle pour trouver la position du caractère entré dans la variable alpha.
5. Écrivez un programme qui demande à l'utilisateur de rentrer une lettre puis qui affiche la lettre qui la précède dans l'alphabet. Indice : utilisez la même technique que dans le problème précédent.
6. Écrivez un programme qui demande à l'utilisateur de rentrer un entier positif et qui affiche la somme des chiffres du nombre entré. Par exemple, si l'utilisateur entre 3582, le programme devrait afficher 18 (3 + 5 + 8 + 2).
7. Ecrivez un programme qui demande à l'utilisateur de rentrer un mot et qui affiche les lettres dans le sens inverse, en utilisant le bloc **dire**.
8. Écrivez un programme qui demande à l'utilisateur de rentrer un nombre et qui insère un espace entre chaque chiffre. Par exemple, si l'utilisateur rentre 1234, la chaîne de sortie devrait être 1 2 3 4. Indice : mettez en place la variable de sortie en regroupant les lettres individuelles du nombre rentré avec des espaces.
9. *Comparaison DeFractions.sb2* Dans ce problème, vous allez créer un jeu qui demande au joueur de comparer des fractions. L'interface utilisateur est présentée ci-après.

Copyright © 2016 Eyrolles.

Lorsqu'on clique sur le bouton Nouveau, le jeu sélectionne aléatoirement deux fractions à comparer. L'utilisateur sélectionne supérieur à (>) ou inférieur à (<) ou égal (=) en cliquant sur le bouton opérateur. Lorsque l'utilisateur clique sur le bouton Vérifier, le jeu contrôle la réponse et fournit un retour. Ouvrez le fichier *ComparaisonDeFractions.sb2* et ajoutez-y les scripts nécessaires pour compléter le jeu.



Copyright © 2016 Eyrolles.

9

LES LISTES

Les programmes que nous avons écrits jusqu'à présent utilisaient des variables ordinaires pour stocker un unique élément. Cependant, de telles variables ne sont pas très utiles lorsque vous voulez stocker un groupe de valeurs, tels que les numéros de téléphones de vos amis, les noms des livres présents dans votre bibliothèque, ou les températures quotidiennes sur un mois.

Par exemple, si vous vouliez que votre programme se souvienne des numéros de téléphones de 20 personnes parmi vos amis, vous devriez utiliser 20 variables ! L'écriture et la maintenance d'un programme contenant 20 variables seraient assez fastidieuses. Dans ce chapitre, nous étudierons un autre type de données, appelé *liste*, qui offre un moyen pratique et confortable de regrouper des valeurs. Voici ce vous allez y apprendre :

- comment créer et manipuler des listes ;
- comment initialiser des éléments individuels au sein d'une liste et y accéder ;

Copyright © 2016 Eyrolles.

- des techniques fondamentales de tri et de localisation d'éléments ;
- comment utiliser des listes pour créer des applications puissantes.

Tout d'abord, j'expliquerai comment établir des listes dans Scratch, quelles commandes utiliser pour les manipuler et comment les remplir avec des données entrées par l'utilisateur. Nous distinguerons ensuite les listes numériques et les opérations courantes qui leur sont appliquées, telles que la recherche du minimum, du maximum et de la valeur moyenne de leurs éléments. Après cela, nous apprendrons un algorithme qui trie les éléments d'une liste. Nous finirons avec plusieurs exemples de programmes qui vous montreront comment les listes sont utilisées dans la vie réelle.

Les listes dans Scratch

Une liste est un contenant dans lequel vous pouvez stocker plusieurs valeurs et y accéder. Vous pouvez voir les listes comme des commodes avec beaucoup de tiroirs, chaque tiroir contenant un élément unique. Lorsque vous créez une liste, vous la nommez, exactement comme vous le feriez avec les variables. Vous accédez ensuite aux éléments individuels grâce à leur position de stockage au sein de cette liste. La figure 9-1, par exemple, illustre une liste nommée listeJours qui stocke les noms des sept jours de la semaine.

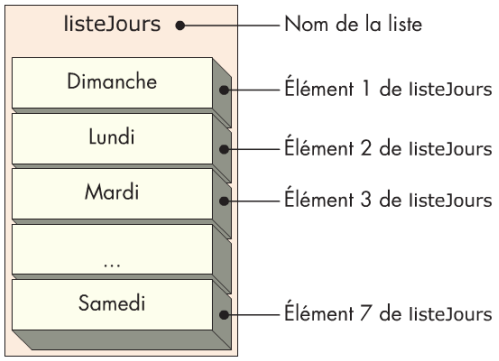


Figure 9-1 : Une liste qui contient les sept jours de la semaine

Vous accédez aux éléments de la liste en utilisant leur *indice de stockage* (ou position). Dans Scratch, le premier élément a un indice de 1, le deuxième élément de 2 et ainsi de suite. Par exemple, puisque Mardi est en troisième position dans la liste, il a un indice de 3. Ainsi, vous accédez au troisième élément de listeJours en utilisant une commande de la forme « élément 3 de listeJours ».

Lançons-nous en créant des listes dans Scratch. Nous verrons les commandes pour gérer et manipuler des listes dans nos programmes et apprendrons comment Scratch réagit aux commandes de listes invalides.

Créer des listes

La création d'une liste est presque identique à celle d'une variable. Sélectionnez la palette *Données* et cliquez sur **Créer une liste** pour faire apparaître la fenêtre de dialogue présentée à la figure 9-2 (à droite). Ensuite, entrez le nom de la liste (nous utiliserons listeJours) et choisissez sa portée. Choisir l'option Pour tous les lutins créera une liste *globale* à laquelle n'importe quel lutin de votre application pourra accéder, alors que l'option Pour

Copyright © 2016 Eyrolles.

ce lutin uniquement crée une liste *locale* qui appartiendra au lutin actuellement sélectionné. Une liste locale ne peut être lue (ou modifiée) que par le lutin qui la possède.

Lorsque vous cliquez sur **OK** pour confirmer votre entrée, Scratch crée une nouvelle liste *vide* et affiche les blocs liés à ce type de variable (figure 9-3). Comme son nom l'indique, une liste vide ne contient aucun élément.

Utilisez ces nouvelles commandes pour manipuler le contenu de votre liste pendant l'exécution de votre script. Vous pouvez ajouter de nouveaux éléments à la suite de la liste, les insérer à une position spécifique, supprimer certains éléments, ou remplacer les valeurs d'éléments existants.



Figure 9-2 : La création d'une liste dans Scratch est similaire à celle d'une variable.

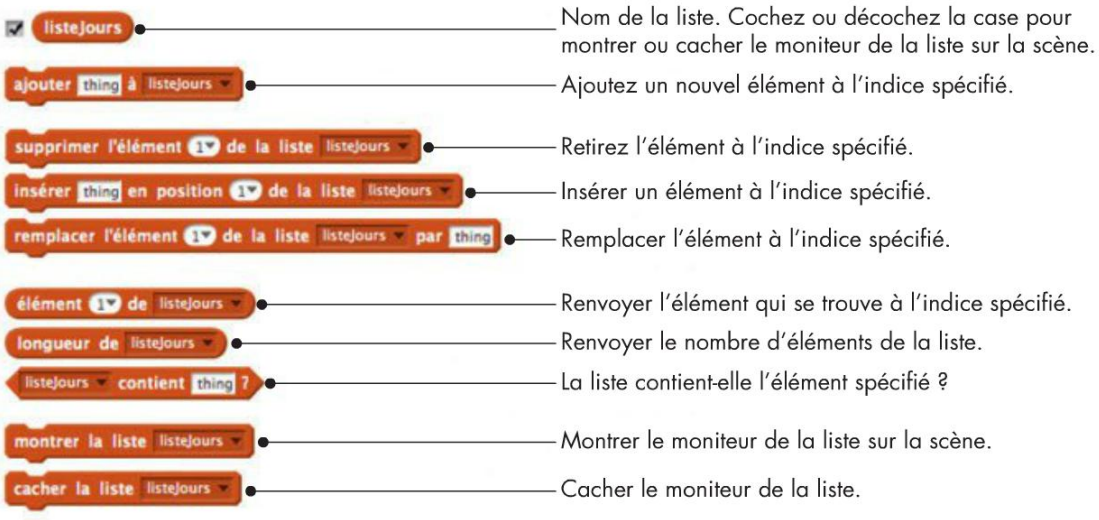


Figure 9-3 : Les blocs de commande et de fonction que vous pouvez utiliser avec des listes

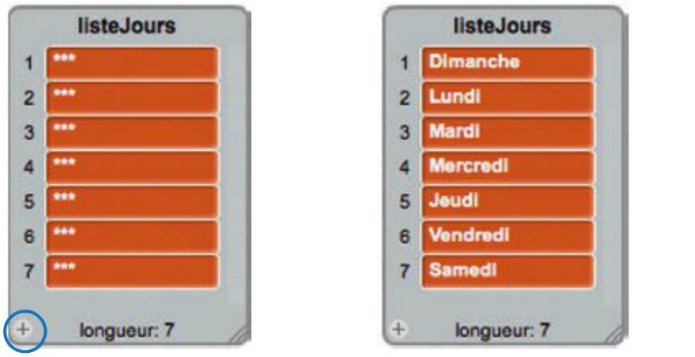
Lorsqu'une nouvelle liste est créée, Scratch en affiche également le moniteur sur la scène (figure 9-4). La liste créée est vide et commence donc avec une longueur de 0. Vous pouvez utiliser le moniteur pour lui ajouter des éléments lors de la création de votre programme.

Copyright © 2016 Eyrolles.



Figure 9-4 : Le moniteur d'une liste qui vient d'être créée est affiché sur la scène.

Si vous connaissez les données que vous voulez stocker dans cette liste (ce qui est le cas de listeJours), vous pouvez les ajouter à votre liste au moment de sa création (figure 9-5).



Cliquez sur le signe plus sept fois pour ajouter un emplacement pour chaque jour.

Cliquez à l'intérieur des cases éditables et entrez vos données.

Figure 9-5 : Remplissage de ListeJours

Cliquez sept fois sur le signe plus dans le coin inférieur droit du moniteur pour créer sept cases et y entrer les jours de la semaine. Utilisez la touche Tabulation ou Entrée pour naviguer entre les différents éléments. Presser une de ces touches met l'élément suivant de la liste en surbrillance et l'édite. Si vous cliquez sur le signe plus lorsqu'un élément est sélectionné, le nouvel élément de la liste sera ajouté après celui-ci. Essayez de naviguer dans la liste !

FAITES LE TEST 9-1

Remplissez listeJours avec les jours de la semaine, comme le montre la figure 9-5.

Les commandes de listes

La figure 9-3 décrit tous les blocs que Scratch a ajoutés lors de la création de listeJours. Dans cette section, nous regarderons de plus près chacun de ces blocs pour mieux en comprendre le fonctionnement.

Copyright © 2016 Eyrolles.

Ajouter et supprimer

La commande **ajouter** place un nouvel élément à la fin de la liste, alors que la commande **supprimer** supprime un élément à l'indice spécifié (figure 9-6).

Le script exécute d'abord la commande **supprimer** pour retirer le deuxième élément de la liste, qui contenait « Orange ». Il ajoute ensuite « Citron » à la fin de la liste grâce à la commande **ajouter**.

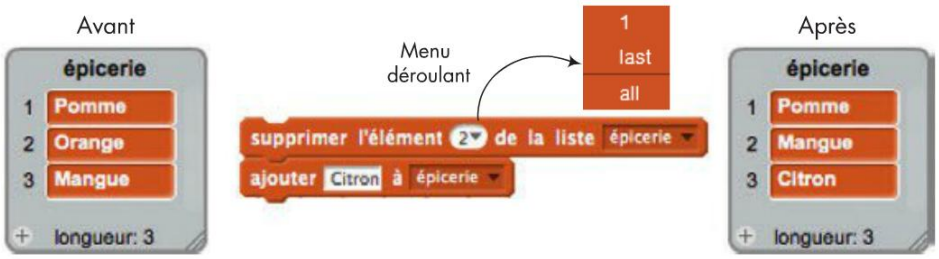


Figure 9-6 : Une liste avant et après l'utilisation des blocs **ajouter** *et* **supprimer**

La commande **ajouter** est triviale, mais examinons la commande **supprimer** de plus près. Vous y entrez l'indice de l'élément à supprimer directement dans la case de paramètre du bloc, ou bien vous cliquez sur la flèche du menu déroulant, qui propose trois options : 1, dernier et tout. Sélectionnez 1 pour supprimer le premier élément (« Pomme ») de la liste, sélectionnez dernier pour supprimer le dernier élément de la liste (« Mangue »), ou sélectionnez tout pour supprimer tous les éléments de la liste.

Insérer et remplacer

Imaginons que vous vouliez stocker les noms et les numéros de téléphone de vos amis en les classant dans l'ordre alphabétique, tout comme la liste de contacts dans votre téléphone. En construisant votre liste, vous devrez insérer les coordonnées de chacun de vos amis à la bonne position. Plus tard, si l'un de vos amis change de numéro de téléphone, vous devrez éditer la liste pour y effectuer ce changement. Les commandes **insérer** et **remplacer** vous aident à accomplir cette tâche (figure 9-7).

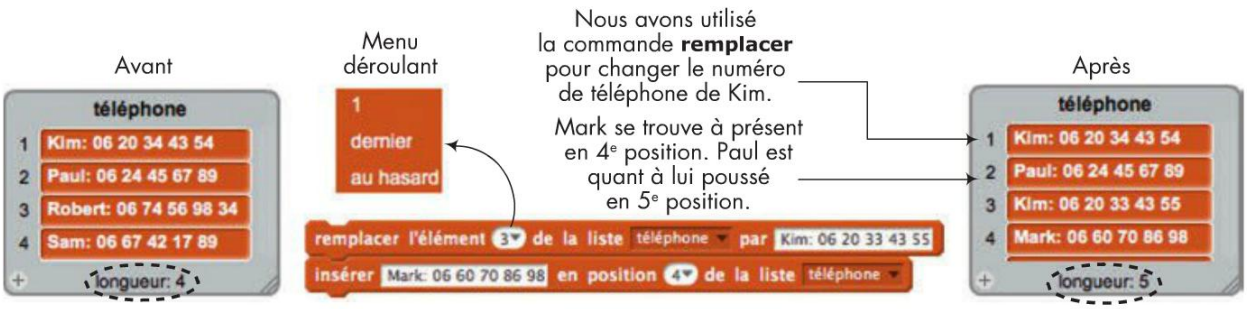


Figure 9-7 : Utilisation des blocs **insérer** *et* **remplacer** *pour mettre à jour une liste de numéros de téléphone*

La commande **remplacer** écrase la chaîne de caractères actuelle de la case numéro 3 par le nouveau numéro de téléphone de Kim. La commande

Copyright © 2016 Eyrolles.

insérer place le numéro de téléphone d'un nouvel ami, Mark, à la case numéro 4 de la liste. Vous remarquerez que les éléments existants sont déplacés d'une case vers le bas pour laisser la place au nouveau numéro entré.

Cliquer sur la petite flèche du numéro de l'élément dans les commandes **remplacer** et **insérer** fait apparaître un menu déroulant contenant trois options : 1, dernier et au hasard (figure 9-7). Si vous sélectionnez au hasard, la commande choisira un élément de façon aléatoire. Vous verrez quelques applications utiles de cette fonctionnalité plus tard dans ce chapitre.

Accéder aux éléments d'une liste

Vous accédez aux éléments d'une liste en utilisant leur indice. Par exemple, le script de la figure 9-8 montre comment utiliser le bloc **élément de** pour accéder aux éléments de listeJours. Le script utilise une variable nommée pos (*position*) pour accéder à chaque élément de la liste en en montrant le contenu grâce à la commande **dire**.

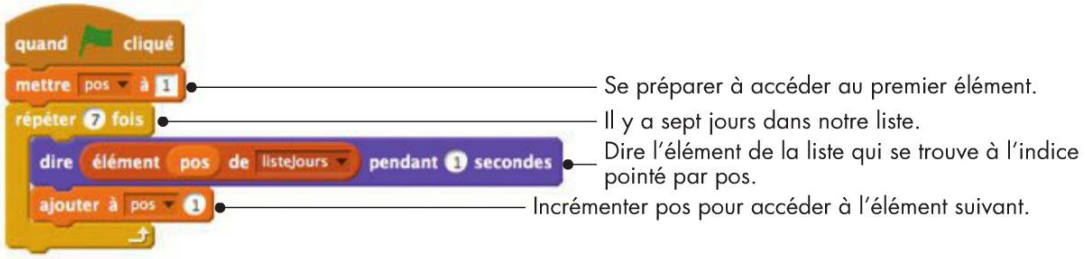


Figure 9-8 : Ce script fait afficher au lutin les sept jours de notre liste listeJours.

Le script initialise la valeur de pos à 1 afin d'accéder au premier élément de listeJours et entre ensuite dans une boucle. Celle-ci effectue 7 répétitions, correspondant au nombre d'éléments dans notre liste. À chaque itération, elle affiche l'élément de la liste qui se trouve à l'indice indiqué par pos et incrémente ce compteur pour accéder à l'élément suivant. En d'autres termes, pos est utilisé comme indice pour pointer sur un élément spécifique de la liste.

FAITES LE TEST 9-2

Remplacez le nombre 7 de la boucle **répéter** par le bloc **longueur de liste-Jours** C'est ce que vous devrez faire pour accéder à tous les éléments d'une liste dont vous ne connaissez pas le nombre d'éléments. Vous pouvez également sélectionner **au hasard** dans le menu déroulant du bloc **élément de**. Cela affichera un élément choisi aléatoirement dans la liste.

Le bloc contient

Vous vérifiez si une chaîne de caractères est présente dans une liste grâce au bloc **contient**, un bloc booléen qui renvoie vrai ou faux si la liste contient ou non la chaîne spécifiée (figure 9-9). Puisque listeJours contient la chaîne de caractères « Vendredi », la commande **dire** à l'intérieur du bloc **si** est exécutée.

Copyright © 2016 Eyrolles.

NOTE *Le bloc* **contient** *ne prend pas en compte les majuscules. Le bloc* **listeJours contient vendREDI**, *par exemple, est évalué comme vrai.*

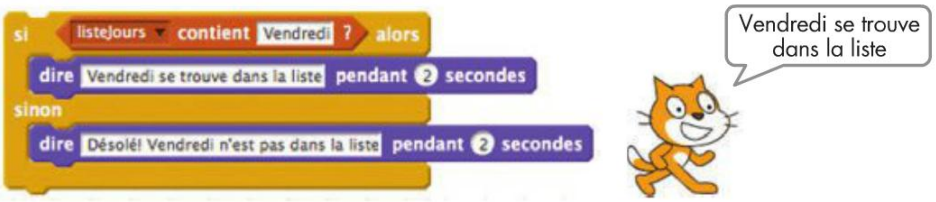


Figure 9-9 : Utilisation du bloc **contient** *pour vérifier si une chaîne de caractères est contenue dans une liste.*

Vérification des limites

Les quatre blocs de listes **supprimer**, **insérer**, **remplacer** et **élément de** nécessitent un paramètre d'entrée qui spécifie l'indice de l'élément auquel vous voulez accéder. Par exemple, pour supprimer le septième élément de listeJours, nous utilisons **supprimer l'élément 7 de listeJours**. À votre avis, que se passerait-il si vous utilisiez un indice invalide avec l'un de ces blocs ? Par exemple, comment réagirait Scratch si vous lui demandiez de supprimer le huitième élément de listeJours (qui n'en contient que sept) ?

Techniquement, le fait d'essayer d'accéder à un élément en dehors des limites d'une liste est une erreur. Plutôt que d'afficher un message d'erreur ou de mettre fin au programme de façon intempestive, Scratch essaiera silencieusement de faire quelque chose de judicieux avec le bloc fautif. C'est pourquoi l'absence de message ne signifie pas qu'il n'y a pas d'erreur. Les problèmes peuvent toujours être présents dans votre code et vous devrez y remédier. Scratch ne vous préviendra pas lorsqu'un indice invalide sera rentré dans un bloc, mais la sortie ne sera généralement pas celle que vous attendiez. Le tableau 9-1 montre ce qui se passe lorsque vous essayez d'accéder à listeJours en utilisant un indice hors limite.

Tableau 9-1 : Résultats inattendus causés par des indices hors limites

Blocs de commande ou de fonction	Résultat
élément 8 de listeJours	Renvoie une chaîne de caractères vide puisque listeJours ne possède que sept éléments. Il se passerait la même chose si l'indice était inférieur à 1.
élément 1.9 de listeJours	Scratch ignore le 9 et renvoie le premier élément de listeJours, « Dimanche ». De même, si vous lui demandiez d'accéder à l'élément 5.3, Scratch renverrait le 5^{e} élément, « Jeudi ».
insérer Nouveaujour en position 10 de la liste listeJours	Scratch ignore cette commande car il essaie d'insérer l'élément entre deux autres éléments (inexistants) de la liste. La liste reste donc inchangée.
insérer Nouveaujour en position 8 de la liste listeJours	Cette commande agit comme la commande **ajouter**. Elle ajoute « Nouveaujour » à la fin de la liste.
supprimer l'élément 8 de la liste listeJours	Cette commande est ignorée (car listeJours ne contient que sept éléments) et la liste reste inchangée.

Copyright © 2016 Eyrolles.

Les exemples du tableau 9-1 montrent que, bien que les blocs de Scratch essaient d'agir judicieusement lorsqu'une entrée est invalide, ils ne le feront pas forcément de la bonne façon. Vous devrez fournir au programme les entrées adéquates pour qu'il fonctionne comme vous le désirez.

Jusqu'à présent, nos exemples ont utilisé de simples listes que nous avons créées manuellement grâce à leurs moniteurs. Comment faire si nous ne connaissons pas à l'avance le contenu d'une liste ? Par exemple, vous pourriez avoir besoin de dresser une liste des nombres entrés par l'utilisateur ou de valeurs aléatoires à chaque exécution du programme. Nous nous attaquerons à ce problème dans la section suivante.

Les listes dynamiques

Les listes sont puissantes car elles peuvent être agrandies ou rétrécies de façon dynamique. Par exemple, imaginons que vous écriviez un programme permettant à un enseignant d'entrer les notes de ses élèves pour les traiter ensuite. L'enseignant pourrait vouloir trouver la note maximale, la minimale, la moyenne, la médiane, etc. Le nombre d'élèves est probablement différent pour chaque classe. Comment indiquer à votre programme quand l'enseignant a terminé de rentrer les notes d'une classe ?

Tout d'abord, nous présenterons deux façons de remplir des listes avec des données entrées par l'utilisateur. Nous étudierons ensuite les listes numériques et verrons quelques opérations qui peuvent leur être appliquées. Une fois que vous en aurez compris les concepts fondamentaux, vous serez prêts à appliquer ces techniques dans vos propres applications.

Remplir des listes avec des entrées utilisateur

Il y a deux manières courantes de remplir une liste avec des données entrées par un utilisateur. Dans la première méthode, votre programme commence par demander à l'utilisateur combien d'entrées lui seront nécessaires, puis démarre une boucle pour obtenir ces données (figure 9-10).

Une fois que l'utilisateur a répondu, le script démarre une boucle dont le nombre de répétitions équivaut au nombre de notes à saisir. Chaque itération de la boucle demande à l'utilisateur de rentrer une nouvelle note et l'ajoute à la fin de la liste.

La deuxième manière de remplir une liste dynamiquement consiste à demander à l'utilisateur de rentrer une valeur spéciale (appelée *sentinelle*) pour indiquer la fin de la liste. Vous devrez bien sûr choisir une sentinelle qui ne peut pas être confondue avec un membre de la liste. Par exemple, si vous attendez une liste de noms ou de nombres positifs, une sentinelle de -1 peut être un bon choix. En revanche, si



Figure 9-10 : Demander à l'utilisateur combien de notes il désire rentrer

Copyright © 2016 Eyrolles.

l'utilisateur doit entrer des valeurs négatives, alors une sentinelle de -1 serait inappropriée. Une sentinelle de -1 fonctionnera pour notre listeDeNotes (figure 9-11).

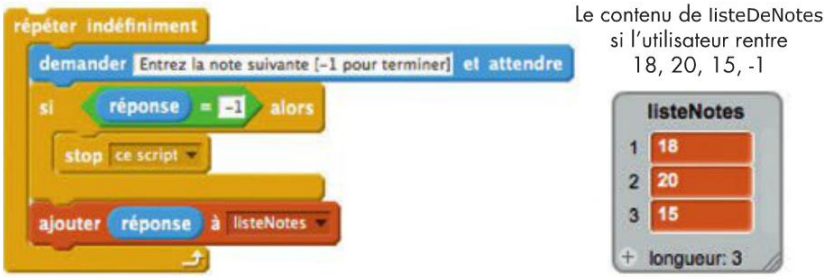


Figure 9-11 : Utilisation d'une sentinelle pour contrôler la fin du remplissage d'une liste

À chaque itération de la boucle, le script demande de rentrer un nombre, qu'il compare à la valeur de la sentinelle. Notez que ce script indique cette dernière dans la question posée à l'utilisateur. Si ce dernier rentre -1, alors le script s'arrête car il sait que l'utilisateur a fini de rentrer ses notes. Sinon, la valeur entrée est ajoutée à la fin de la liste et le programme redemande à l'utilisateur de rentrer une note. La figure 9-11 montre à quoi listeDeNotes devrait ressembler si l'utilisateur avait rentré trois notes avant la sentinelle.

Création d'un histogramme

Histogramme .sb2

Je vous propose comme exemple pratique l'écriture d'une application qui dessine un *histogramme* (un graphique dont les données sont représentées par des barres) à partir de données entrées par l'utilisateur. Pour assurer la simplicité, nous n'accepterons que cinq nombres compris entre 1 et 40. Une fois que le programme aura reçu ces cinq nombres, il dessinera cinq barres dont la hauteur sera proportionnelle aux valeurs entrées (figure 9-12).

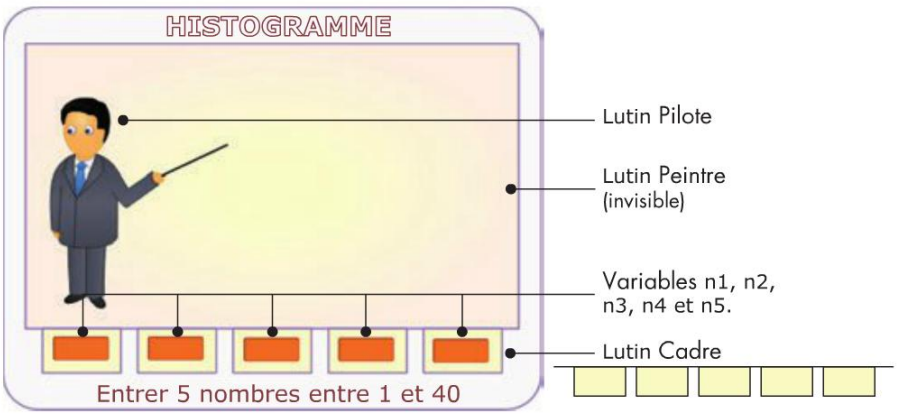


Figure 9-12 : L'application Histogramme

Cette application contient trois lutins. Pilote contrôle le déroulement de l'application ; il contient des scripts qui reçoivent les entrées de l'utilisateur, remplissent la liste et indiquent au lutin Peintre de commencer à dessiner. Peintre est un lutin invisible qui dessine l'histogramme. Le lutin Cadre est purement décoratif ; il cache le bas de chaque barre de l'histogramme pour

Copyright © 2016 Eyrolles.

les faire apparaître plates ; sans lui, le bas des barres verticales serait arrondi. Les valeurs numériques des cinq barres sont affichées grâce à cinq variables, nommées n1 à n5, dont les moniteurs sont situés aux positions adéquates sur la scène. Lorsque vous cliquez sur le drapeau vert pour démarrer l'application, le lutin Pilote démarre le script de la figure 9-13.

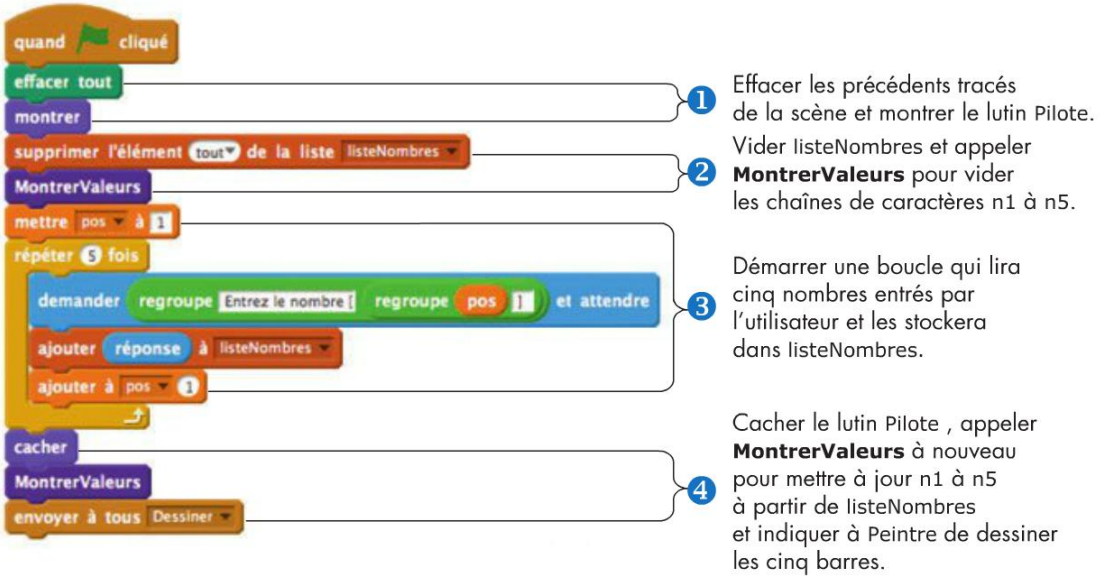


Figure 9-13 : Le script principal du lutin Pilote

Tout d'abord, le lutin Pilote apparaît sur la scène et efface tout précédent tracé ❶. Le script vide ensuite listeNombres avant de l'utiliser pour collecter les nouvelles entrées de l'utilisateur, puis il appelle **MontrerValeurs** ❷ pour faire en sorte que n1 à n5 aient des moniteurs vides (car listeNombres est vide).

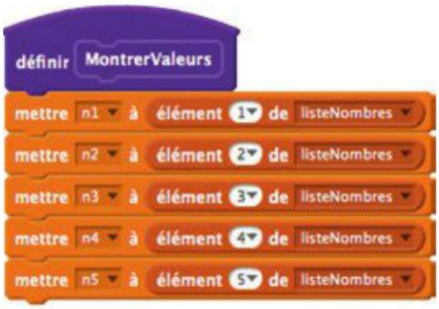


Figure 9-14 : La procédure **MontrerValeurs**

Lorsque la scène est prête, le script démarre une boucle **répéter** ❸, qui se répétera 5 fois. Dans la boucle, le lutin Pilote demande à l'utilisateur de rentrer un nombre et l'ajoute à la fin de listeNombres. Après avoir récupéré et stocké les cinq nombres dans listeNombres, Pilote se cache ❹ pour laisser la place à l'histogramme. Il appelle ensuite de nouveau **MontrerValeurs** pour mettre à jour n1 à n5 avec les nouvelles valeurs et envoie le message Dessiner pour indiquer au Peintre qu'il doit dessiner les cinq barres.

Avant d'étudier comment le lutin Peintre dessine les barres, jetons un œil à la procédure **MontrerValeurs** (figure 9-14).

MontrerValeurs met simplement les variables n1 à n5 aux valeurs correspondantes dans listeNombres. Puisque le premier appel de **MontrerValeurs** est passé immédiatement après avoir vidé listeNombres, toutes les variables contiendront initialement une chaîne de caractères vide. Voilà pourquoi les cinq moniteurs seront vides sur la scène, ce qui correspond exactement à ce

Copyright © 2016 Eyrolles.

que nous voulons. Lorsque listeNombres contient des données entrées par l'utilisateur, l'appel de **MontrerValeurs** affiche les données dans ces mêmes moniteurs.

Étudions maintenant la procédure **Dessiner** qui s'exécute lorsque le lutin Peintre reçoit le message Dessiner (figure 9-15).

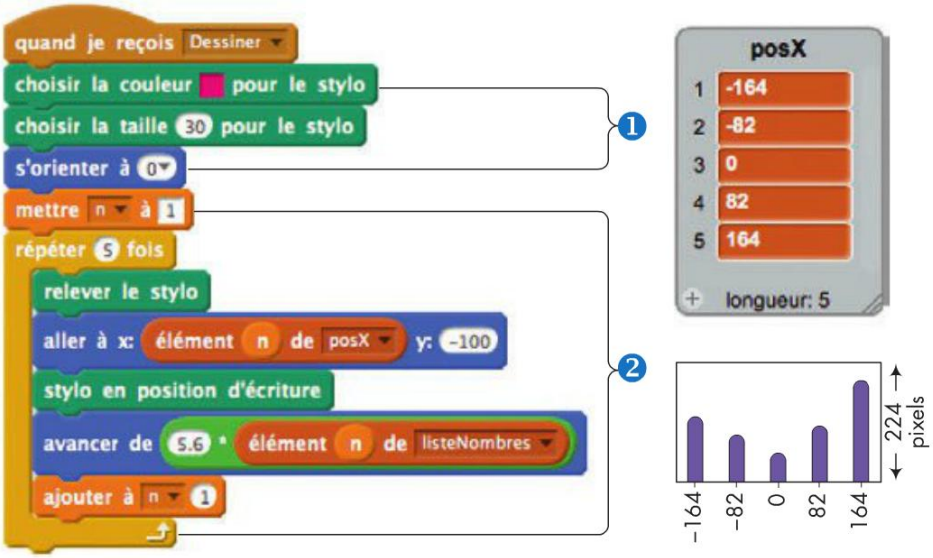


Figure 9-15 : Le script **Dessiner** *du lutin* Peintre

Tout d'abord, le lutin initialise sa couleur. Il règle ensuite la taille de son stylo à une valeur élevée afin de dessiner des barres épaisses. Pour se préparer à dessiner des barres verticales, le lutin s'oriente vers le haut ❶.

Le script démarre une boucle **répéter** pour dessiner les cinq barres ❷. Nous connaissions la position *x* de chaque barre à l'avance ; nous avons donc créé une liste nommée posX pour stocker ces valeurs (également présentées sur la figure 9-15). À chaque itération de la boucle, le lutin Peintre se déplace à la position *x* de la barre actuelle, pose son stylo en position d'écriture, puis se déplace vers le haut pour dessiner une ligne verticale.

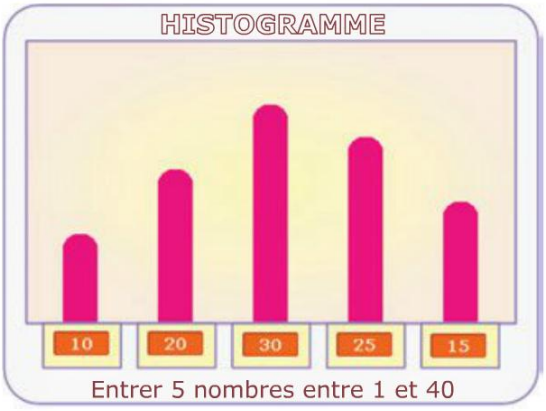


Figure 9-16 : Un exemple de résultat de l'application

La hauteur de chaque ligne est proportionnelle à la valeur correspondante dans listeNombres. La zone de notre graphique sur la scène a une hauteur de 224 pixels et, puisque 40 est la valeur d'entrée maximale, une entrée de 40 devrait mener à une barre dont la hauteur atteint le haut de la zone du graphique. Pour trouver la hauteur (en pixels) d'un nombre de listeNombres, nous devons multiplier ce nombre par 5,6 (ce qui correspond

Copyright © 2016 Eyrolles.

à 224/40). La figure 9-16 montre le résultat de l'application après avoir obtenu des données de l'utilisateur. Notez que le lutin Cadre couvre les bouts arrondis du dessin du stylo pour que le bas des barres semble plat.

FAITES LE TEST 9-3

Exécutez cette application plusieurs fois pour comprendre comment elle fonctionne. Changez le script pour faire en sorte que chaque barre soit dessinée d'une couleur différente. Indice : créez une nouvelle liste, nommée couleur, pour que le lutin Peintre stocke la valeur de la couleur des cinq barres et utilise la commande suivante pour dessiner chaque barre.



Listes numériques

Les listes de nombres apparaissent dans de nombreuses applications pratiques. Elles servent à tester des notes, des mesures de températures, le prix de produits, etc. Dans cette section, nous étudierons quelques opérations classiques qui vous serviront sur les listes numériques. Nous écrirons plus particulièrement des procédures trouvant la valeur maximum ou minimum et la moyenne des nombres d'une liste.

Trouver le maximum et le minimum

TrouverMaximum .sb2

Supposons que vous soyez un(e) enseignant(e) et que vous cherchiez à connaître la meilleure note du dernier contrôle de la classe. Vous pourriez écrire un programme (figure 9-17) qui compare toutes les notes d'une liste et trouve la maximale.

La procédure **TrouverMaximum** démarre en affectant à noteMax la valeur de la première note de la liste. Elle démarre ensuite une boucle pour comparer les autres nombres de la liste avec la valeur actuelle de noteMax. À chaque fois qu'une valeur supérieure à noteMax est trouvée, elle est affectée à noteMax. Lorsque la boucle prend fin, noteMax contient la meilleure note de la liste.

Trouver la valeur minimale d'une liste suit un algorithme similaire. Il faut commencer par supposer que le premier élément de la liste est le minimum, puis utiliser une boucle pour le comparer aux éléments restants. À chaque fois qu'une valeur plus faible est trouvée, elle remplace celle de la variable qui contient le minimum.

Copyright © 2016 Eyrolles.

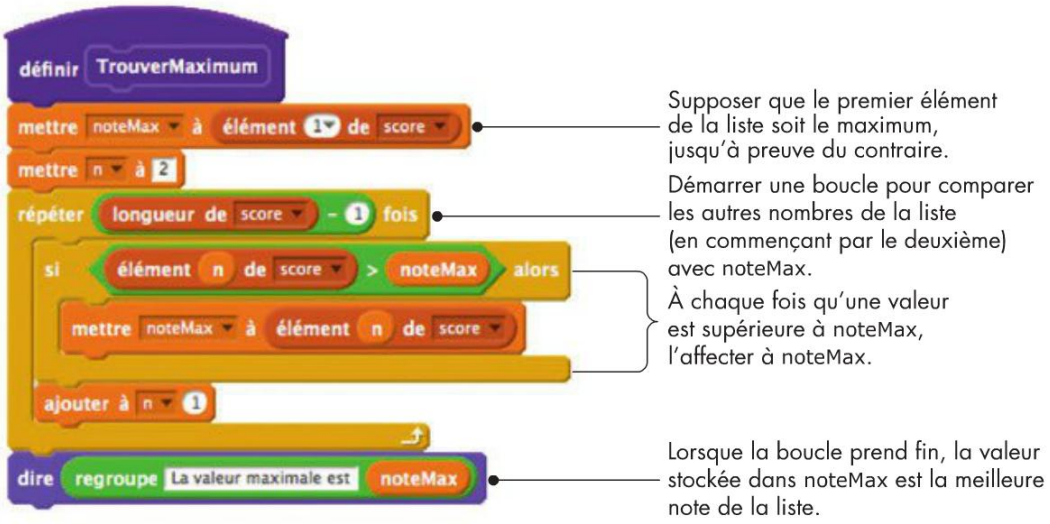


Figure 9-17 : Trouver la valeur maximale d'une liste

FAITES LE TEST 9-4

Utilisez ce que vous avez appris dans cette section pour créer une procédure appelée **TrouverMinimum** qui trouvera la valeur minimale d'une liste de notes.

Trouver la moyenne

TrouverLaMoyenne.sb2

Dans notre prochain exemple, nous écrirons une procédure qui calcule la moyenne des *N* notes rentrées dans la liste, en calculant d'abord leur somme puis en divisant le résultat par *N* (figure 9-18).

La procédure **TrouverMoyenne** utilise une boucle pour accéder à toutes les notes de la liste, en faire la somme et stocker le résultat dans une variable nommée somme. Cette variable est initialisée à 0 avant de démarrer la boucle. Lorsque la boucle prend fin, le script calcule la moyenne en divisant somme par le nombre de notes et stocke le résultat dans une variable nommée moyenne.

NOTE *Portez une attention particulière à la façon dont nous avons fait évoluer la variable* somme *dans la boucle. Ce modèle, connu sous le nom modèle de stockage, est très utilisé en programmation.*

Copyright © 2016 Eyrolles.

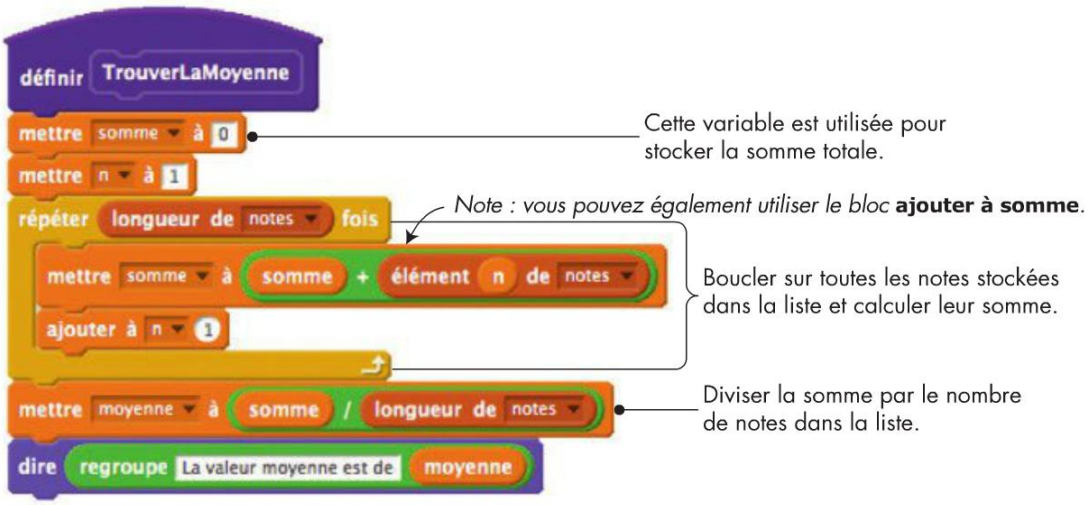


Figure 9-18 : Trouver la valeur moyenne d'une liste de nombres

Localiser et trier les éléments d'une liste

Imaginons que vous ayez une liste de contacts qui ne soit pas rangée dans un ordre particulier. Si vous voulez organiser les contacts, vous devrez les *trier* selon l'ordre alphabétique de leurs noms. Si vous cherchez le numéro de téléphone d'une personne dont vous connaissez le nom de famille, vous devrez *localiser* ce nom de famille et vérifier s'il se trouve dans votre liste de contacts ou non. Le but de cette section est de vous présenter les techniques de programmation de base pour la localisation et le tri des éléments d'une liste.

Recherche linéaire

Localisation.sb2

Le bloc **contient** de Scratch fournit une manière simple de vérifier si une liste contient un élément spécifique. Si, en plus de cela, nous voulons connaître la position de cet élément, alors nous devons effectuer la recherche nous-même.

Cette section présente une méthode pour localiser des éléments d'une liste appelée *recherche linéaire* (ou *recherche séquentielle*). Simple à comprendre et à implémenter, elle fonctionne sur n'importe quelle liste, triée ou non. Cependant, puisqu'une recherche linéaire compare les valeurs recherchées à tous les éléments de la liste, cela peut prendre un temps assez long, selon la longueur de la liste.

Imaginons que vous cherchiez un élément spécifique dans une liste nommée fruits. Si la liste contient l'élément que vous recherchez, vous devez également connaître sa position exacte dans la liste. La procédure **Localiser** présentée à la figure 9-19 effectue une recherche linéaire dans la liste fruits pour nous donner la réponse que nous recherchons.

Copyright © 2016 Eyrolles.

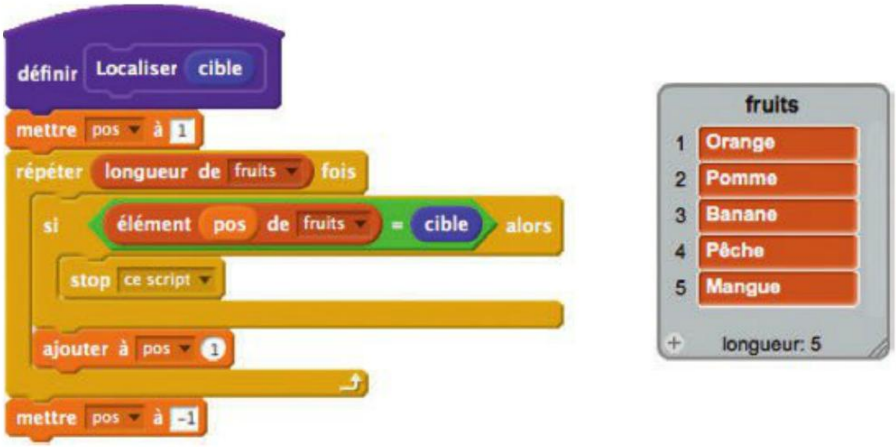


Figure 9-19 : La procédure **Localiser**

En commençant par le premier élément, **Localiser** compare un par un les fruits de notre liste avec celui spécifié dans le paramètre cible. La procédure s'arrête si la valeur cible a été trouvée ou bien si la fin de la liste a été atteinte. Si le script trouve la réponse que nous voulons, la variable pos contient la position de l'élément trouvé. Sinon, la procédure affecte à pos une valeur invalide (-1 dans ce cas) pour indiquer que l'élément cible ne se trouve pas dans la liste. La figure 9-20 montre un exemple de script appelant la procédure et utilisant son résultat.



Figure 9-20 : Utilisation de la procédure **Localiser**

L'évaluation de la valeur de pos indique deux choses au script appelant : (a) si l'élément que nous recherchons se trouve dans la liste et (b) la position exacte à laquelle il se trouve, s'il s'y trouve. L'exécution de ce script affecte 4 à pos, indiquant que « Pêche » a été trouvé à la quatrième position dans la liste fruits.

Fréquence d'occurrence

ComptagedEléments.sb2

Supposons que votre école ait mis en place un sondage sur la qualité de la nourriture de sa cafétéria. Les élèves devaient donner une note comprise entre 1 = mauvais et 5 = excellent. Chaque vote a été introduit dans une liste et vous avez



Figure 9-21 : La procédure **RemplirListe**

Copyright © 2016 Eyrolles.

été embauché pour écrire un programme qui traitera ces données. Pour le moment, l'école veut uniquement savoir combien d'élèves détestent complètement la nourriture (c'est-à-dire combien d'élèves ont donné une note de 1).

Il est clair que votre programme devra contenir une procédure qui compte combien de fois le nombre 1 apparaît dans la liste. Pour simuler les votes des élèves, utilisons une liste appelée sondage qui contient 100 valeurs aléatoires comprises entre1 et 5 (figure 9-21).

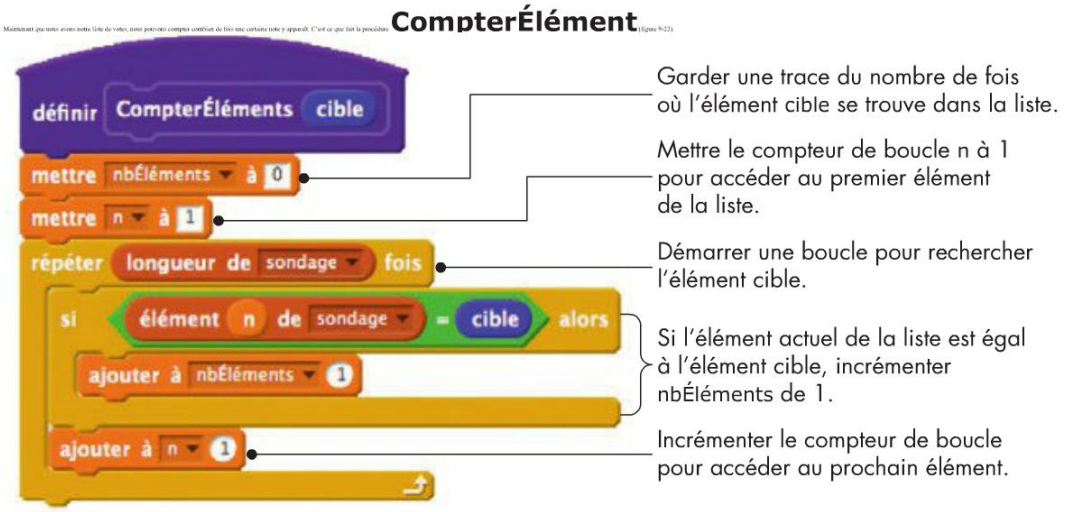


Figure 9-22 : Comptage du nombre de fois qu'un certain élément apparaît dans une liste

Le paramètre cible représente l'élément à rechercher et la variable nbÉléments correspond au nombre de fois où il a été trouvé. La procédure démarre en mettant nbÉléments à 0 avant de rentrer dans une boucle **répéter** pour rechercher la valeur spécifiée par cible dans la liste. À chaque itération de la boucle, la procédure contrôle l'élément de la liste se trouvant à la position indiquée par n, le compteur de boucle. Si cet élément est égal à la cible, le script incrémente nbÉléments de 1.

Pour savoir combien d'élèves n'apprécient pas du tout la nourriture de l'école, nous devons simplement appeler **CompterÉléments** avec un argument de 1 (figure 9-23).

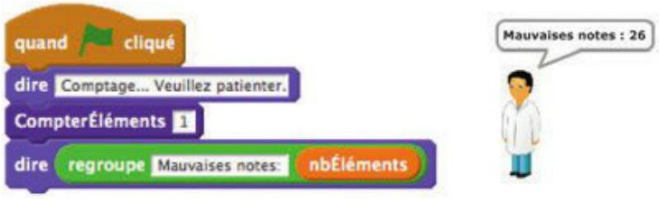


Figure 9-23 : Utilisation de la procédure **CompterÉléments**

Copyright © 2016 Eyrolles.

Tri à bulle

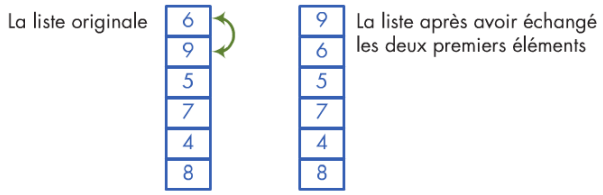
TriÀBulles.sb2

Si vous disposez d'un ensemble de noms, de scores d'un jeu, ou de quoi que ce soit d'autre que vous voudriez faire apparaître dans un certain ordre – alphabétique, croissant ou décroissant, etc. –, vous devez trier votre liste. Il existe de nombreuses manières de procéder. Le *tri à bulle* fait partie des algorithmes de tri les plus simples. Son nom se réfère à la manière dont les valeurs remontent la liste jusqu'à leur bonne position, telle une bulle dans un tube.

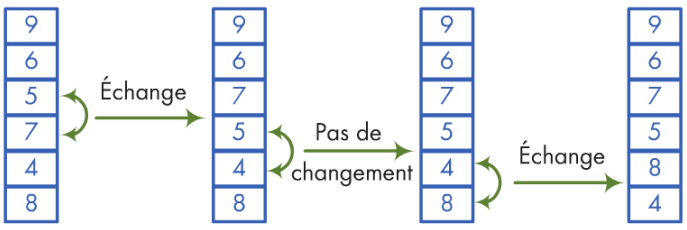
> **FAITES LE TEST 9-6**
>
> Pour compléter ce sondage, la direction de l'école a souhaité également connaître le nombre d'élèves qui adorent la nourriture de la cafétéria, ainsi que le nombre d'élèves qui ont participé au sondage. Modifiez le programme et exécutez-le à nouveau pour fournir ces nouvelles informations.

Imaginons que vous ayez besoin de trier la liste de nombres [6 9 5 7 4 8] dans l'ordre décroissant. Les étapes suivantes illustrent le fonctionnement de l'algorithme de tri à bulles.

1. Nous commençons par comparer les deux premiers éléments de la liste. Puisque 9 est plus grand que 6, nous pouvons échanger leur position.



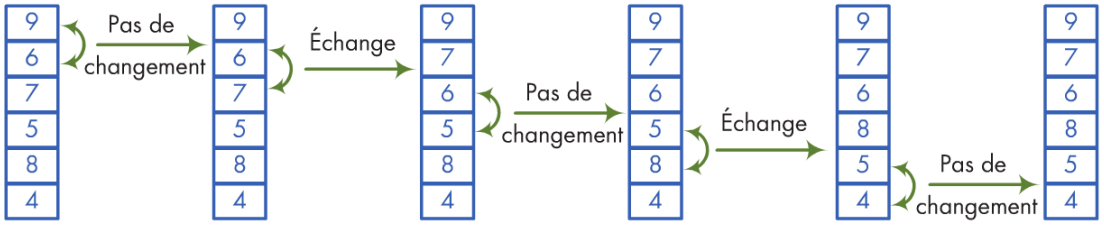
2. Nous pouvons maintenant comparer le deuxième élément avec le troisième. Puisque 6 est supérieur à 5, les deux nombres sont déjà dans l'ordre et nous pouvons passer à la paire suivante.
3. Nous répétons ensuite ce procédé pour comparer le troisième et le quatrième éléments, le quatrième et le cinquième, puis finalement le cinquième et le sixième. Observez ci-après la liste après ces trois comparaisons.



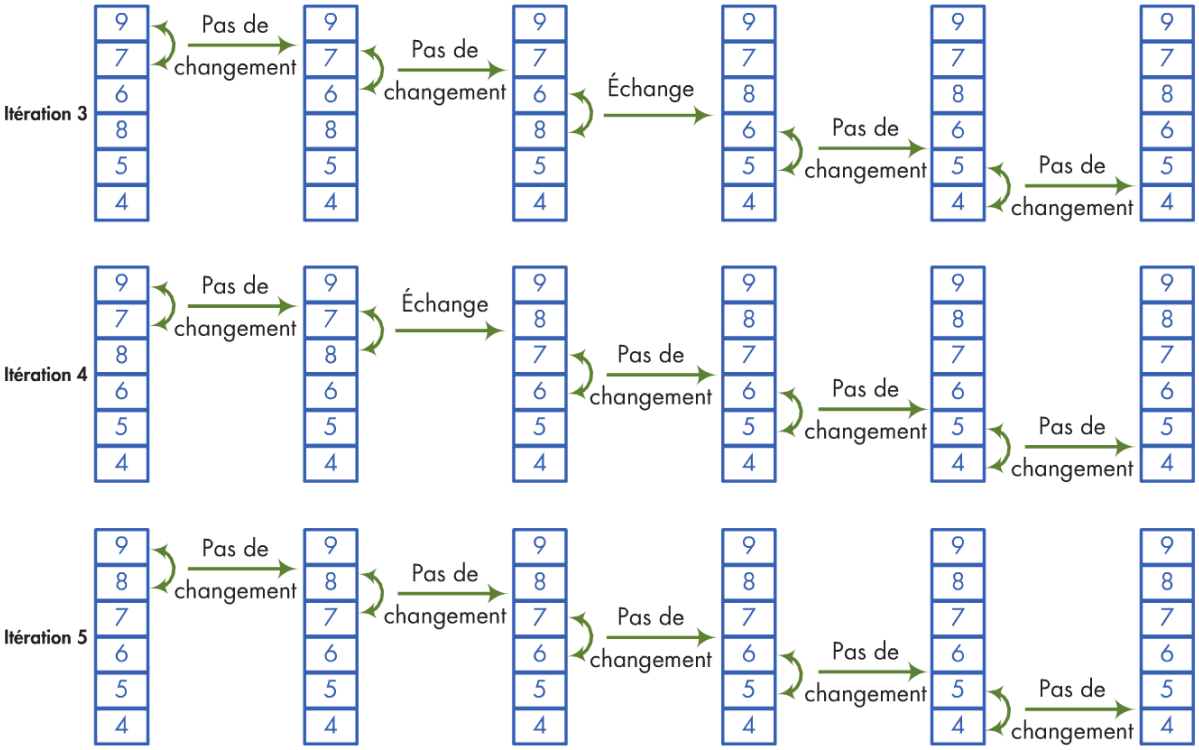
4. La première itération du tri à bulle est terminée, mais notre liste n'est toujours pas dans le bon ordre. Nous devons donc effectuer une seconde

Copyright © 2016 Eyrolles.

itération, en recommençant à la première étape. Une fois de plus, nous allons comparer les éléments par paires et les échanger au besoin. Voici la liste après la seconde itération :



5. Nous répétons le procédé de tri à bulle jusqu'à ce qu'il n'y ait plus d'échange de nombres lors d'une itération, ce qui signifie que notre liste est triée. Les trois dernières itérations de l'algorithme sont présentées ci-après.



Maintenant que vous savez comment fonctionne le tri à bulle, implémentons-le dans Scratch. Le script présenté à la figure 9-24 possède deux boucles. La boucle interne parcourt une liste, en comparant et en échangeant les éléments au besoin et met un drapeau (nommé fait) à 0 lorsqu'une autre itération est nécessaire. La boucle externe se répète tant que le drapeau fait est à 0, puisque cette valeur signifie que le tri n'a pas encore été effectué. Si la boucle interne accomplit une itération sans échanger d'élément, la boucle externe prend fin, ainsi que la procédure.

Explorons cette procédure un peu plus en détail. Puisque nous n'avons pas encore commencé à trier, elle met fait à 0 ❶. La boucle externe utilise un bloc **répéter jusqu'à** pour parcourir la liste jusqu'à ce qu'elle soit triée (c'est-à-dire jusqu'à ce que fait passe à 1) ❷. Au début de chaque itération,

Copyright © 2016 Eyrolles.

la boucle met fait à 1 ❸ (nous supposons qu'aucun échange ne sera réalisé). Elle initialise également pos à 1 pour commencer le tri par le premier élément de la liste.

La boucle interne compare ensuite chaque paire d'éléments dans la liste. La boucle doit effectuer $N-1$ comparaisons ❹, avec N le nombre d'éléments de la liste.

Si l'élément se trouvant à la position pos+1 est supérieur à celui se trouvant à pos ❺, ils doivent être échangés. Sinon, la procédure ajoute 1 à pos afin de comparer la paire d'éléments suivante. L'échange d'éléments est effectué grâce à une variable temporaire nommée temp ❻.

Une fois que l'itération prend fin, la boucle interne remet fait à 0 si des éléments ont été échangés ou laisse fait=1 si aucun changement n'a été effectué ❼. La boucle externe continue jusqu'à ce que la liste soit triée.

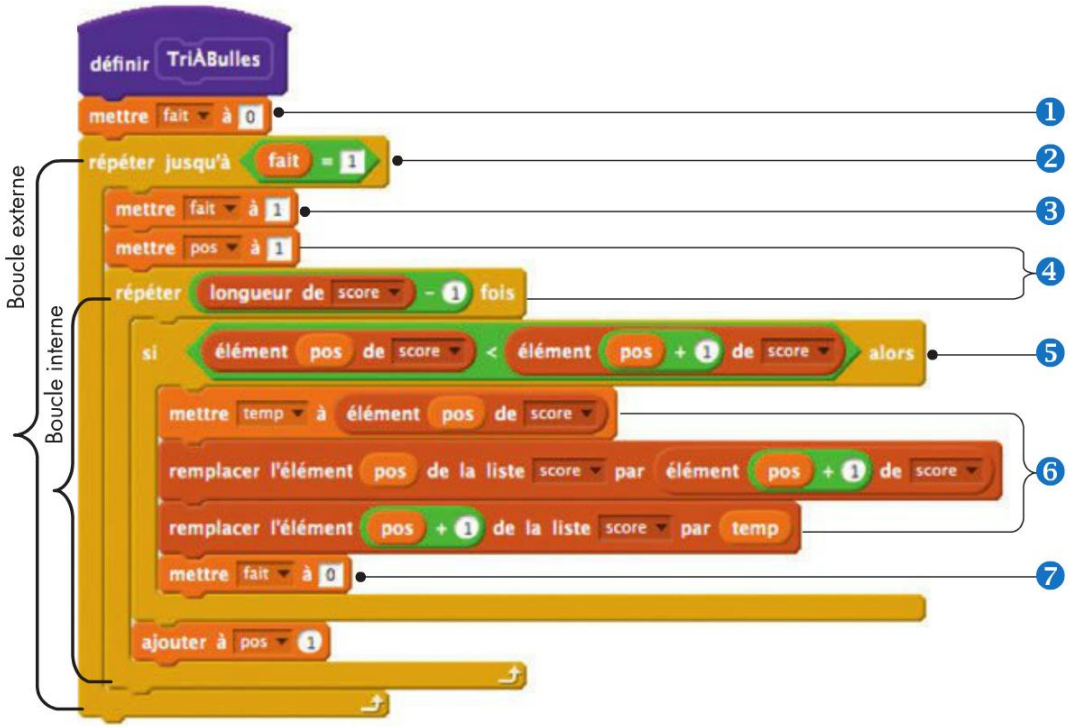


Figure 9-24 : La procédure **TriÀBulles**

Trouver la médiane

Médiane.sb2

Maintenant que nous savons comment trier une liste, nous pouvons facilement trouver la valeur médiane d'une séquence de nombres. Pour rappel, la médiane est la valeur centrale d'en ensemble de nombres triés. Si cet ensemble contient un nombre pair de nombres, la médiane est la moyenne des deux nombres centraux. Nous pouvons décrire la médiane d'une liste triée contenant N éléments de la manière suivante :

Copyright © 2016 Eyrolles.

FAITES LE TEST 9-7

Créez une liste de noms (et non de nombres) et utilisez le script de tri à bulle pour l'ordonner. Cela fonctionne-t-il toujours comme il le faudrait ? Quels changements devez-vous appliquer à la procédure pour qu'elle trie dans l'ordre croissant ?

$$médiane = \begin{cases} \text{élément à l'indice } \dfrac{N+1}{2} \text{ si } N \text{ est impair} \\ \text{moyenne des éléments à } \dfrac{N}{2} \text{ et } \dfrac{N}{2}+1 \text{ si } N \text{ est pair} \end{cases}$$

Une procédure qui effectue ce calcul est présentée à la figure 9-25. Elle suppose que la liste est triée.

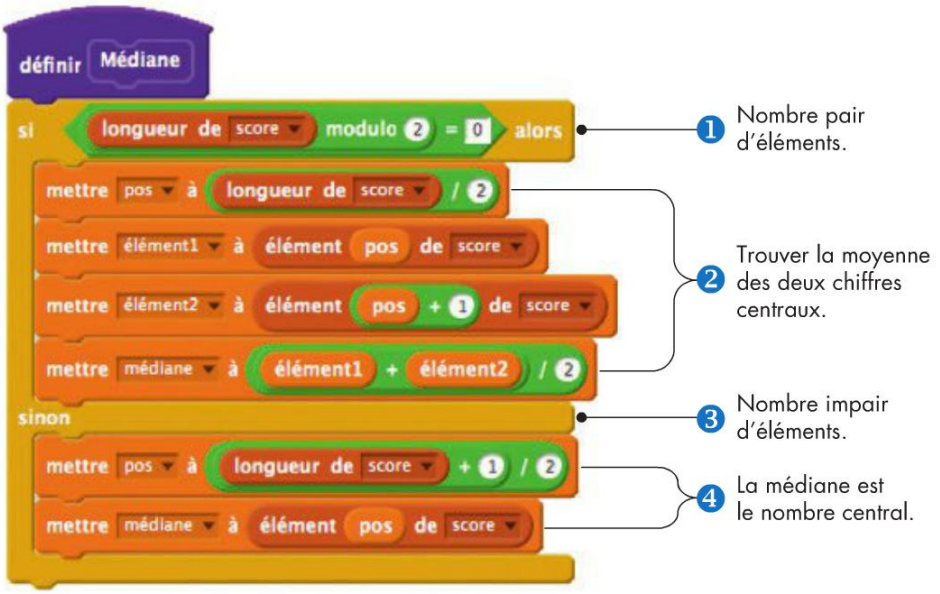


Figure 9-25 : Trouver la valeur médiane d'une liste de nombres triés

La procédure utilise un bloc **si/sinon** pour gérer les cas où les listes contiennent un nombre pair ou impair d'éléments. Si la division par 2 du nombre d'éléments n'a pas de reste (donc si la liste a un nombre pair d'éléments) ❶, la variable médiane se voit affecter la valeur moyenne des deux nombres centraux ❷. Sinon, la liste possède un nombre impair d'éléments ❸ et la variable médiane se voit affecter la valeur du nombre central de la liste ❹.

Il est maintenant temps d'appliquer les nombreux concepts que nous avons vus à quelque chose de plus stimulant. Dans le reste de ce chapitre, nous verrons plusieurs exemples qui vous montreront comment utiliser les listes dans des applications plus complexes.

Copyright © 2016 Eyrolles.

Projets Scratch

Dans cette section, vous étudierez des projets Scratch pratiques qui mettront en évidence différents aspects des listes. Je présenterai également de nouvelles idées et techniques que vous pourrez utiliser dans vos propres créations.

Le poète

Poète.sb2

Le premier projet de ce chapitre est un générateur de poèmes. Notre poète artificiel sélectionne des mots aléatoirement à partir de cinq listes (articles, adjectifs, noms, verbes et prépositions) et les combine selon un modèle prédéfini. Pour donner un thème central à notre poème, tous les mots de ces listes sont liés à l'amour et à la nature. Nous finirons sans doute avec des poésies idiotes, mais faisons-le pour le plaisir !

NOTE *L'idée de ce programme est adaptée du livre* Learning with Logo *de Daniel Watt (McGraw-Hill, 1983). Vous trouverez la liste de mots complète que nous avons utilisée dans le fichier Scratch de ce projet, Poète.sb2.*

Chaque poème est composé de trois lignes qui suivent trois modèles :

- vers 1 : article, adjectif, nom ;
- vers 2 : article, nom, verbe, préposition, article, adjectif, nom ;
- vers 3 : nom, adjectif, adjectif.

Avec ces structures en tête, regardons la procédure qui construit le premier vers du poème (figure 9-26).

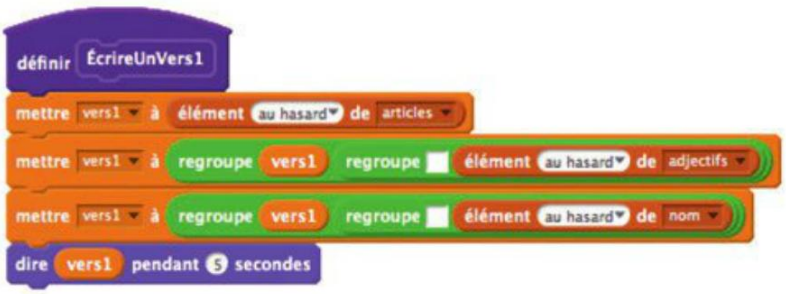


Figure 9-26 : « Écriture » du premier vers d'un poème

Ce script sélectionne un mot aléatoire de la liste articles et le stocke dans vers1. Le script y ajoute ensuite un espace, un mot aléatoire de la liste adjectifs, un autre espace et un mot aléatoire de la liste noms. Enfin, le lutin Poète énonce le vers complet. Je ne montrerai pas les procédures des deux autres vers du poème car elles sont extrêmement similaires, mais vous pouvez ouvrir *Poète.sb2* pour les voir.

Voici deux poèmes créés par notre programme :

- Chaque route prestigieuse
- Un poisson avance derrière chaque maison
- Un étang calme et bleu

Copyright © 2016 Eyrolles.

- Toute goutte glacée
- Un cœur fixe sous toute porte effrayante
- Reine timide et silencieuse

NOTE *Par souci de simplicité, nous n'avons pas pris en compte les règles d'accord de nombre et de genre.*

Jeu de classification de quadrilatères

ClassifQuad .sb2

Notre prochain projet est un jeu simple qui vous permettra d'étudier les différentes sortes de quadrilatères. Le jeu affiche l'une des six formes (parallélogramme, losange, rectangle, carré, trapèze ou cerf-volant) sur la scène et demande à l'utilisateur de classer cette forme en cliquant sur le bouton adéquat (figure 9-27).

FAITES LE TEST 9-8

Poète.sb2

Ouvrez *Poète.sb2* et exécutez-le plusieurs fois pour découvrir les capacités d'écriture de notre poète virtuel. Changez ensuite ce programme pour qu'il utilise trois lutins, chacun étant responsable d'une ligne du poème, ce qui vous permettra de lire le poème en entier sur la scène d'un seul coup.

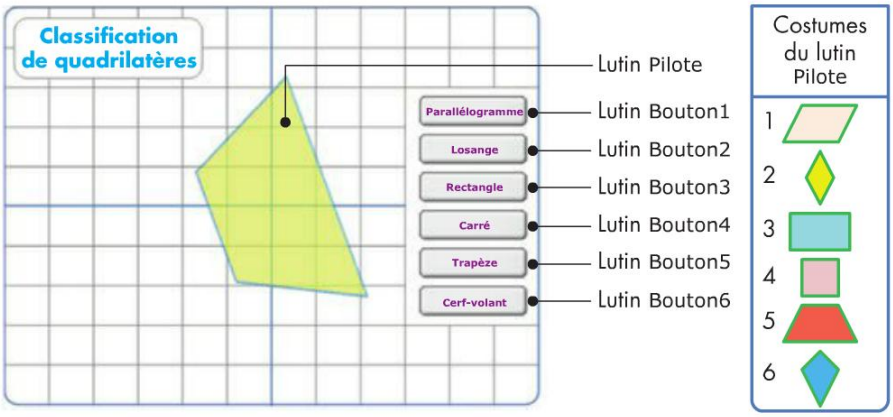


Figure 9-27 : L'interface utilisateur du jeu de classification de quadrilatères

Le jeu contient sept lutins : six pour les boutons de réponse et un septième (nommé Pilote) qui contient le script principal. Comme le montre la figure 9-27, le lutin Pilote possède six costumes qui correspondent aux six quadrilatères du jeu. Lorsqu'on clique sur le drapeau vert, Pilote exécute le script présenté à la figure 9-28.

Tout d'abord, le lutin Pilote est mis au premier plan ❶ afin qu'il ne soit pas caché par un bouton. Dans la boucle principale du jeu ❷, le script affiche

Copyright © 2016 Eyrolles.

un quadrilatère choisi aléatoirement à chaque passage dans **MontrerForme** ❸. Après avoir affiché le quadrilatère, le script initialise la variable globale choix à 0 pour indiquer que l'utilisateur n'a pas encore répondu ❹.

Le script attend ensuite ❺ que choix devienne non nul, ce qui arrive lorsque le joueur clique sur l'un des six boutons. Il appelle alors **VérifierRéponse** ❻ pour indiquer au joueur si sa réponse est correcte.

Maintenant que vous savez comment le script principal fonctionne, jetons un œil à la procédure **MontrerForme** (figure 9-29).

Tout d'abord, **MontrerForme** déplace le lutin Pilote au centre de la scène et l'oriente dans une direction aléatoire ❶. Il affecte à la variable forme une valeur aléatoire comprise entre 1 et 6 et change le costume du lutin ❷ pour afficher un quadrilatère que le joueur devra identifier.

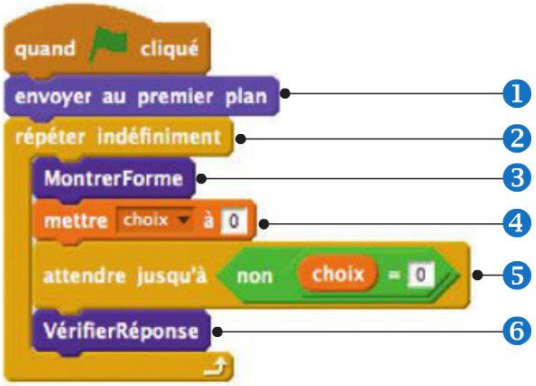


Figure 9-28 : Le script principal du lutin Pilote

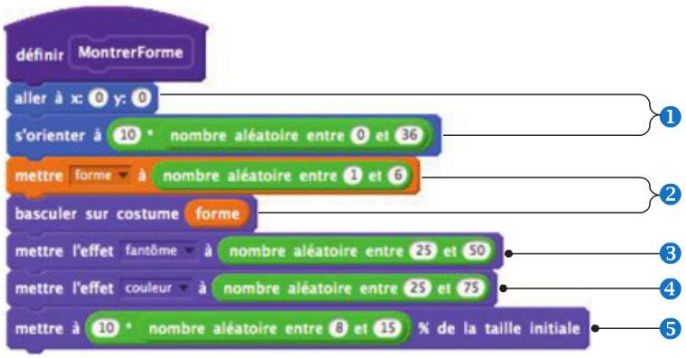


Figure 9-29 : La procédure **MontrerForme** *du lutin* Pilote

Pour que la grille de l'arrière-plan reste visible, **MontrerForme** règle le niveau de transparence ❸ à une valeur aléatoire comprise entre 25 et 50. Pour donner l'impression qu'une nouvelle forme apparaît à chaque tour, la procédure règle également l'effet couleur à une valeur aléatoire pour changer la couleur du costume ❹ et redimensionne le lutin à 80 %, 90 % … ou 150 % de sa taille originale ❺.

Voyons ensuite brièvement les scripts des six lutins boutons, présentés à la figure 9-30. Ils sont identiques, sauf pour les valeurs affectées à la variable choix.



Figure 9-30 : Scripts des lutins boutons

Copyright © 2016 Eyrolles.

Ces scripts d'une ligne règlent chacun la valeur de choix à un nombre différent dépendant du bouton sur lequel le joueur a cliqué. Une fois que choix contient la réponse du joueur, la procédure **VérifierRéponse** (figure 9-31) peut comparer cette variable avec la valeur de forme, qui spécifie le type de quadrilatère dessiné.

Si choix et forme sont égaux, alors la réponse du joueur est correcte. Sinon, la réponse est fausse et le lutin annonce la forme correcte. **VérifierRéponse** utilise la variable forme comme indice pour une liste nommée nomQuad (figure 9-31) pour obtenir le nom correct de la forme affichée.

FAITES LE TEST 9-9

ClassifQuad .sb2

Ouvrez *ClassifQuad.sb2* et exécutez-le plusieurs fois pour comprendre comment il fonctionne. Tel qu'il est écrit, le jeu ne s'arrête jamais. Modifiez le programme pour y ajouter un critère de fin de jeu. Gardez également une trace du nombre de réponses fausses et correctes du joueur.

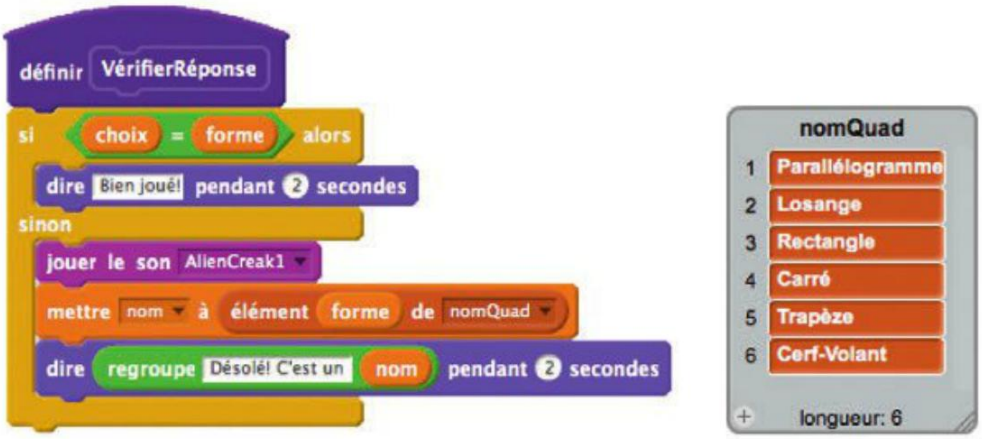


Figure 9-31 : La procédure **VérifierRéponse**

Magicien des mathématiques

Magicien Mathématique.sb2

Cette application montre deux manières de rendre les listes encore plus utiles. Nous verrons comment les utiliser pour stocker des ensembles de données non uniformes (c'est-à-dire des ensembles de tailles différentes) et également comment utiliser les valeurs d'une liste comme indices d'une autre liste. Dans notre cas, chaque ensemble de données contient la réponse à une énigme ainsi que les questions menant à cette réponse. Bien que chaque énigme ne contienne qu'une seule réponse, le nombre d'instructions varie d'une énigme à l'autre.

Copyright © 2016 Eyrolles.

Notre magicien des mathématiques demande à l'utilisateur de penser à un nombre « secret » et effectue une séquence d'opérations sur ce nombre (doubler le nombre, soustraire 2, diviser la réponse par 10, etc.). À la fin, une fois que le joueur aura effectué toutes les opérations, le magicien utilise ses pouvoirs magiques pour énoncer à l'utilisateur le résultat qu'il a trouvé sans même connaître le nombre originellement choisi. Le tableau 9-2 illustre le fonctionnement de ce jeu.

Tableau 9-2 : Fonctionnement du Magicien des mathématiques

Instructions du Magicien	Votre Nombre
Pensez à un nombre.	2
Ajoutez 5.	7
Multipliez par 3.	21
Soustrayez 3.	18
Divisez par 3.	6
Soustrayez votre nombre original.	4

Après la dernière instruction, le magicien vous dira que le résultat des opérations que vous avez effectuées est 4, bien que le jeu ne sache pas que vous aviez commencé avec 2. Essayez cette énigme avec différents nombres pour comprendre l'astuce du magicien !

L'interface de l'application est présentée à la figure 9-32.

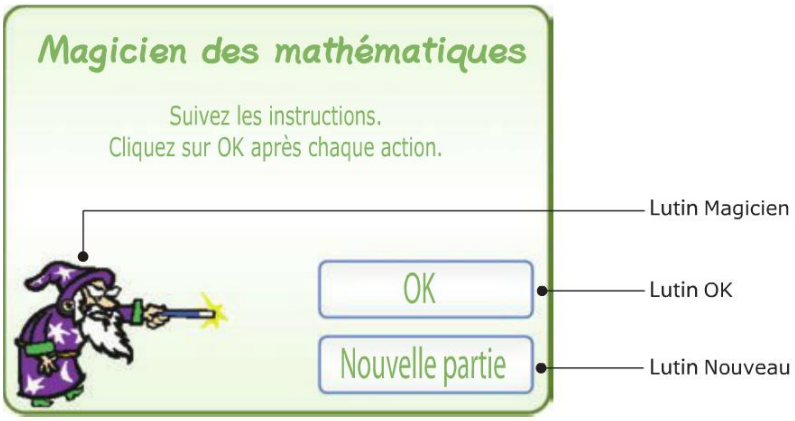


Figure 9-32 : L'interface utilisateur de l'application Magicien des mathématiques

L'application contient trois lutins : Magicien, qui donne les instructions au joueur, OK et Nouveau, correspondant aux boutons de même nom. Elle utilise également deux listes illustrées à la figure 9-33.

Copyright © 2016 Eyrolles.

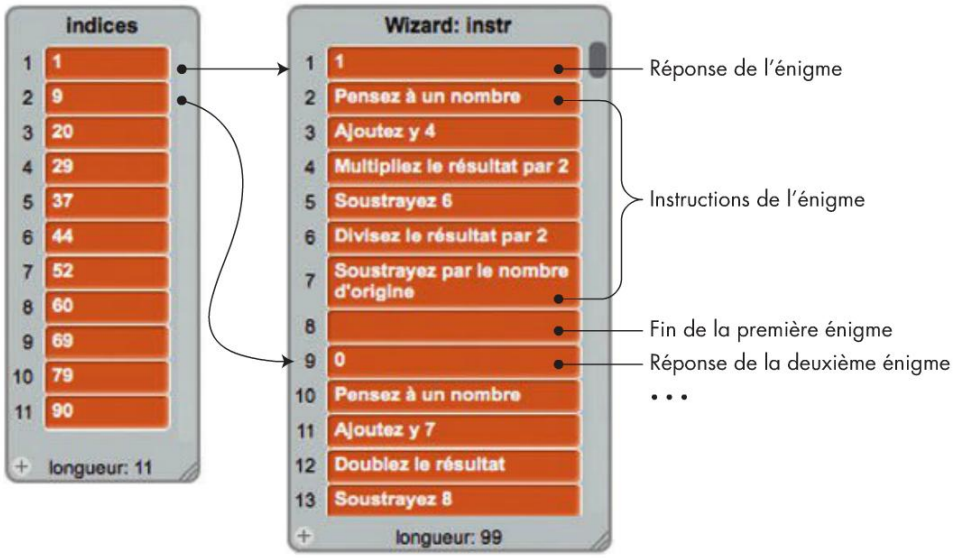


Figure 9-33 : Les deux listes utilisées par le lutin Magicien

La liste instr (à droite) contient 11 énigmes. Chaque ensemble de données inclut (a) la réponse à l'énigme, (b) les instructions et (c) un élément vide pour signaler la fin de l'ensemble. Les données de la liste de gauche (nommée indices) identifient l'indice de départ de chaque énigme dans la liste instr. Par exemple, le deuxième élément de la liste indices est 9, ce qui signifie que l'ensemble de données de la deuxième énigme démarre à la neuvième position de la liste instr (figure 9-33). Voici la description d'une stratégie pour le développement de ce jeu :

1. Lorsque l'utilisateur démarre une nouvelle partie, sélectionner un nombre aléatoire compris entre 1 et 11 (puisque notre jeu contient actuellement 11 énigmes).
2. Consulter la liste indices pour trouver la position de départ de l'ensemble des données de l'énigme sélectionnée. Par exemple, si la deuxième énigme est sélectionnée, la liste indices indique que l'ensemble des données de cette énigme démarre à l'indice 9 dans la liste instr.
3. Accéder, dans la liste instr, à l'indice trouvé à l'étape 2. L'élément se trouvant à cet indice est interprété comme la réponse de l'énigme. Les éléments suivants représentent les instructions du magicien.
4. Faire déclamer au magicien les différentes instructions une par une jusqu'à rencontrer l'élément vide, qui indique la dernière instruction. Le magicien doit attendre que l'utilisateur appuie sur le bouton OK avant d'énoncer une nouvelle instruction.
5. Dévoiler la réponse de l'énigme.

Maintenant que nous savons comment le jeu doit fonctionner, regardons les scripts des deux boutons (figure 9-34).

Copyright © 2016 Eyrolles.



Figure 9-34 : Le script des deux lutins boutons

Lorsqu'on clique sur le bouton Nouvelle Partie, il envoie le message NouvellePartie. Lorsqu'on clique sur le bouton OK en réponse à une instruction, le lutin met le drapeau cliqué à 1 pour indiquer au lutin Magicien que le joueur a fini de suivre ses instructions. Lorsque Magicien reçoit le message NouvellePartie, il exécute le script de la figure 9-35.

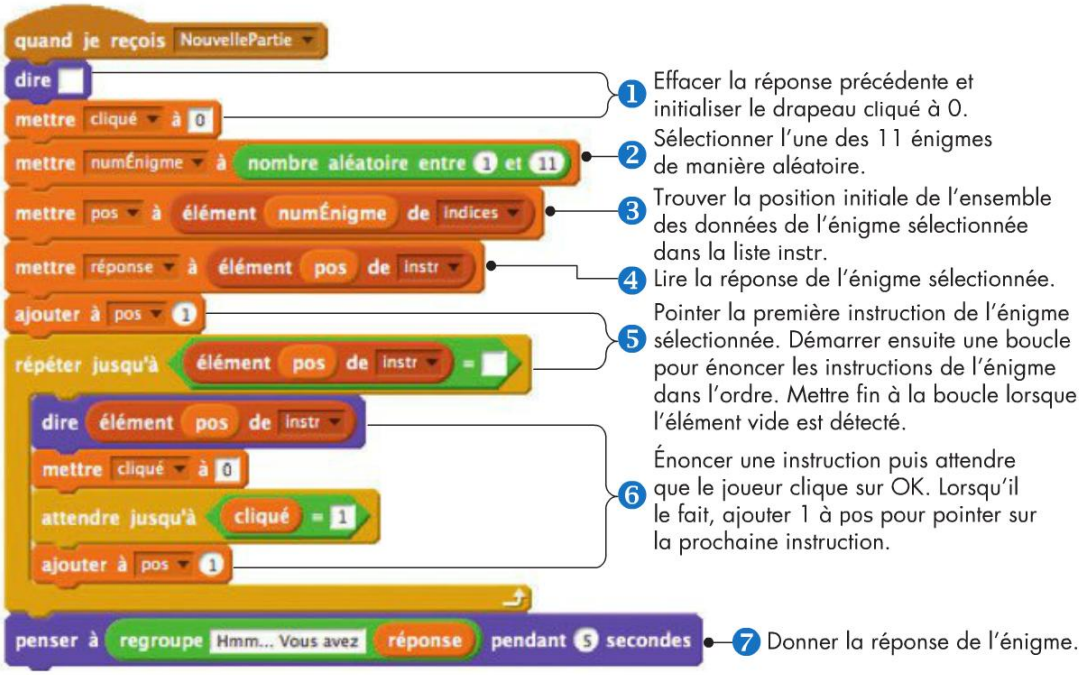


Figure 9-35 : Le script NouvellePartie *du lutin* Magicien

NouvellePartie démarre en effaçant la bulle de bande dessinée de l'énigme précédente (s'il y en avait) et initialise le drapeau cliqué à 0 ❶. Le script stocke ensuite un nombre aléatoire dans la variable numÉnigme ❷, correspondant au numéro de l'énigme qui sera énoncée. Après cela, il lit la position initiale de l'ensemble des données de l'énigme, indiquée par la liste indices, et la stocke dans la variable pos ❸. Le script utilise pos pour lire la réponse de l'énigme, qu'il stocke dans réponseÉnigme ❹. Ensuite, le script ajoute 1 à pos pour pointer sur la première instruction et démarre une boucle **répéter jusqu'à** pour énoncer les instructions dans l'ordre ❺. Le script attend que le drapeau cliqué soit mis à 1 avant de passer à l'instruction suivante ❻. La boucle prend fin lorsqu'elle détecte un élément vide et le script donne la réponse de l'énigme ❼.

Copyright © 2016 Eyrolles.

Quizz sur l'anatomie d'une fleur

AnatomieFleur .sb2

Dans cette section, nous utiliserons un quizz sur les différentes parties d'une fleur pour montrer comment construire des quizz simples dans Scratch. La figure 9-36 montre l'interface de notre application au début du jeu et après que le programme a vérifié les réponses de l'utilisateur. Pour répondre, il faut entrer les lettres à côté du label adéquat puis cliquer sur le bouton Vérifier. Le programme compare alors les entrées de l'utilisateur avec les réponses correctes et indique les bonnes et les mauvaises en utilisant des symboles verts et rouges, respectivement.

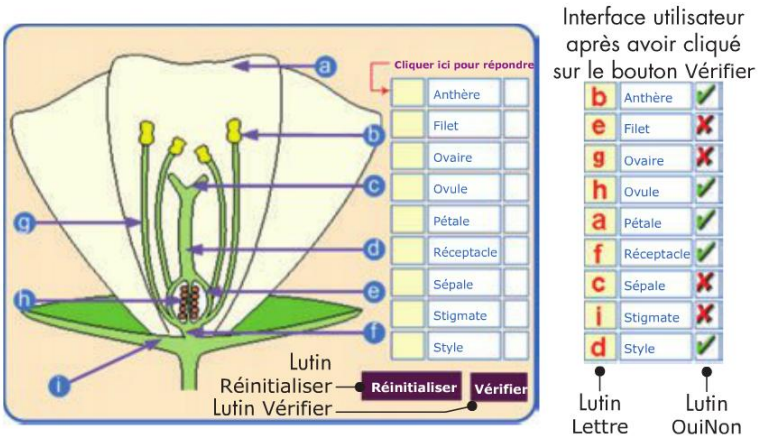


Figure 9-36 : Interface utilisateur du quizz sur l'anatomie d'une fleur

Ce quizz utilise trois listes. La première (nommée repCorrectes) contient les caractères qui correspondent aux bonnes réponses des neuf parties du quizz. La deuxième liste (nommée rep) contient les entrées utilisateur et la troisième liste (nommée centreYCase) contient 11 positions verticales utilisées par les lutins Lettre et OuiNon (afin qu'ils sachent où estampiller leurs costumes). Lorsque l'utilisateur clique sur l'un des emplacements réponse, la scène détecte le clic de la souris et demande à l'utilisateur d'entrer une réponse. La scène met à jour l'élément correspondant de la liste rep pour qu'il corresponde à l'entrée utilisateur et estampille la lettre correspondante au niveau de l'emplacement réponse sélectionné. Ouvrez *AnatomieFleur.sb2* pour voir le script qui lit et affiche les réponses de l'utilisateur.

FAITES LE TEST 9-10

Si vous supprimez l'une des énigmes ou changez le nombre d'instructions de certaines d'entre elles, alors vous devrez reconstruire la liste indices pour qu'elle corresponde à la liste instr. Écrivez une procédure qui remplisse automatiquement la liste indices selon le contenu actuel de instr. Il vous faudra rechercher les chaînes de caractères vides, puisqu'elles indiquent la fin d'un ensemble de données et le démarrage du suivant.

Copyright © 2016 Eyrolles.

Lorsque l'utilisateur clique sur le bouton Vérifier, le lutin OuiNon, qui possède les costumes des symboles verts et rouges, exécute le script présenté à la figure 9-37.

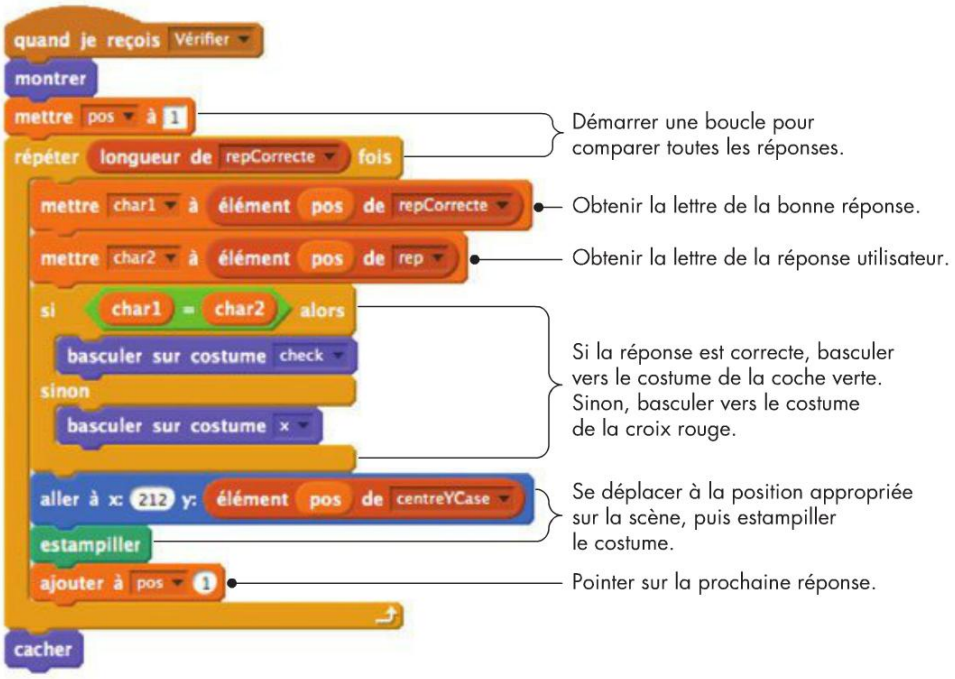


Figure 9-37 : La procédure **Vérifier** *du lutin* OuiNon

Le script compare les éléments des listes repCorrectes et rep un par un. Si les deux valeurs sont égales, il estampille une coche verte. Sinon, il estampille une croix rouge pour indiquer une mauvaise réponse. Quoi qu'il arrive, **Vérifier** consulte la liste centreYCase afin d'obtenir la bonne position pour estampiller l'image.

Autres applications

SayThatNumber.sb2

Vous pouvez télécharger les applications bonus à partir des ressources web de ce livre. Vous y trouverez trois applications à étudier par vous-même, avec les explications complètes (en anglais). La première application est un jeu à deux joueurs de tri de fractions et de nombres décimaux. Chaque joueur possède 5 cartes tirées aléatoirement dans un jeu de 31 cartes. Ensuite, les joueurs recevront une carte supplémentaire, qu'ils pourront décider de garder ou non. Si un joueur garde une carte, il devra la faire glisser au-dessus d'une carte de sa main pour la remplacer par la nouvelle. Le premier joueur à avoir rangé ses cinq cartes dans l'ordre croissant gagne la partie.

SortEmOut.sb2

La seconde application est un programme qui écrit en toutes lettres des nombres entiers. Elle demande à l'utilisateur d'entrer un nombre puis affiche le nombre orthographié. Par exemple, si l'utilisateur entre 3 526, le programme dira « trois mille cinq cent vingt-six. » L'idée est de casser le nombre, de droite à gauche en groupes de trois chiffres. Chaque groupe est ensuite écrit avec un mot multiplicateur (mille, million, etc.) si besoin.

Copyright © 2016 Eyrolles.

Sleve.sb2 Le troisième programme utilise le crible d'Ératosthène, un algorithme permettant de trouver tous les nombres premiers inférieurs à 100.

Résumé

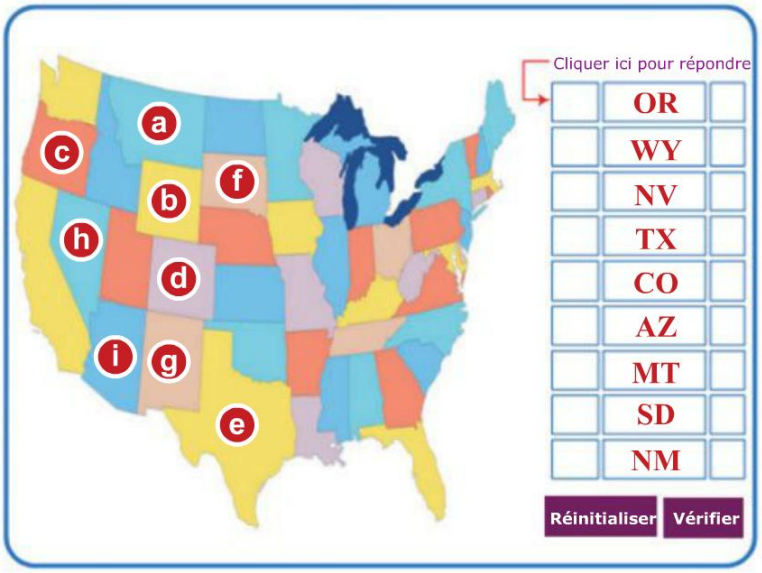
Les listes sont très utiles en programmation et elles fournissent une manière pratique de stocker plusieurs éléments. Dans ce chapitre, nous avons vu comment créer des listes dans Scratch, ainsi que les commandes à utiliser pour les gérer et les remplir dynamiquement avec des données entrées par l'utilisateur.

Nous avons également étudié les listes numériques et montré comment y trouver les valeurs minimale, maximale, moyenne et médiane. Nous avons ensuite appris des algorithmes simples pour trier et localiser des éléments dans une liste. Nous avons conclu ce chapitre avec plusieurs programmes montrant comment utiliser des listes dans des applications pratiques.

QuizzCarteUSA.sb2

FAITES LE TEST 9-11

Ouvrez cette application et essayez-la. Imaginez ensuite d'autres quizz sur différents sujets que vous pourrez créer et implémenter. Un exemple, présenté ci-après, est fourni dans le fichier *quizzCarteUSA.sb2*. Ouvrez ce fichier et complétez les parties manquantes pour faire fonctionner le quizz.



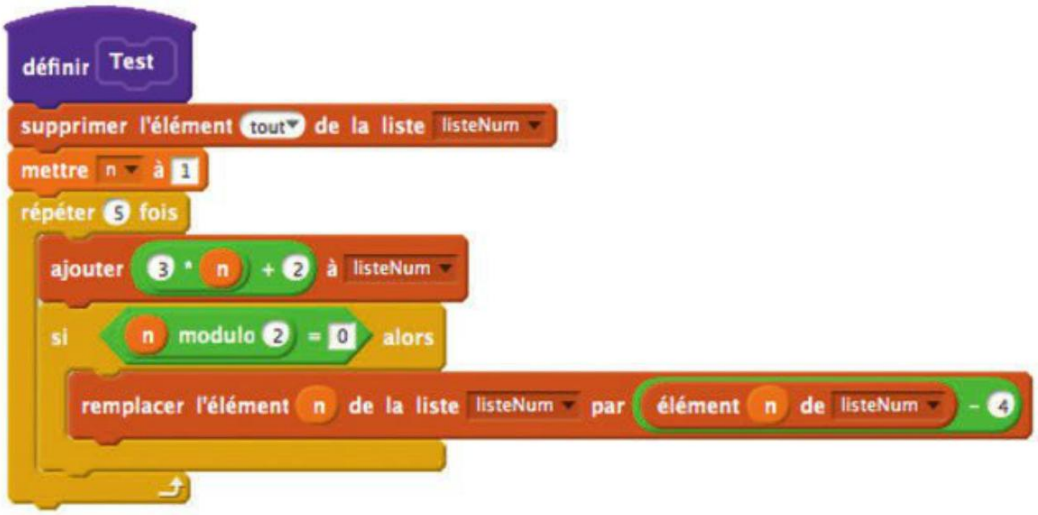
Problèmes

1. Créez une liste qui contient les 10 premiers nombres premiers. Écrivez un script qui affiche ces nombres grâce au bloc **dire**.
2. Créez trois listes pour stocker des données personnelles. La première liste stockera des noms, la deuxième des dates de naissance et la troisième

Copyright © 2016 Eyrolles.

des numéros de téléphone. Écrivez un programme qui demande à l'utilisateur le nom de la personne dont il veut ajouter les informations. Si ce nom est déjà présent dans la première liste, le programme affichera la date de naissance et le numéro de téléphone du contact.

3. Créez deux listes pour stocker les éléments vendus dans une épicerie ainsi que les prix correspondants. Écrivez un programme qui demande à l'utilisateur de rentrer le nom d'un élément et qui affichera le prix correspondant si cet élément se trouve dans la liste.
4. Que contiendra la liste listeNum après avoir exécuté les scripts présentés ci-après ? Recopiez cette procédure et exécutez-la pour vérifier votre réponse.



5. Écrivez un programme qui double chaque élément stocké dans une liste numérique.
6. Écrivez un programme qui demande à l'utilisateur de rentrer des noms d'élèves ainsi que leur moyenne et qui stocke ces données dans deux listes. Arrêtez l'enregistrement des données lorsque l'utilisateur rentre -1 comme nom d'élève.
7. Écrivez un programme qui demande à l'utilisateur de rentrer les températures maximales et minimales pour les 12 mois de l'année. Stockez les valeurs entrées dans deux listes.
8. Écrivez un programme qui demande à l'utilisateur de rentrer 10 entiers. Stockez chaque nombre entré dans une liste seulement s'il n'y est pas déjà contenu.
9. Écrivez un programme qui traite une liste de 20 notes et qui trouve le nombre d'élèves dont la note est comprise entre 15 et 18.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

ANNEXE

PARTAGE ET COLLABORATION

Comme vous allez le découvrir dans cette annexe, Scratch facilite la collaboration et le partage de votre travail avec des gens du monde entier. Vous apprendrez notamment à :

- créer un compte ;
- utiliser votre sac à dos pour travailler avec des lutins et des scripts créés par d'autres utilisateurs ;
- remixer des projets d'autres personnes ;
- publier votre travail et le partager avec la communauté Scratch.

Créer un compte Scratch

Bien que vous n'ayez pas besoin de créer un compte pour utiliser Scratch, le fait d'en avoir un offre certains avantages. Cela vous permet de sauvegarder votre travail sur le site de Scratch, de communiquer avec d'autres utilisateurs et de partager votre travail en ligne. Suivez les étapes ci-après pour créer un compte Scratch.

Copyright © 2016 Eyrolles.

1. Allez à l'adresse *http://scratch.mit.edu/* et cliquez sur le lien **Rejoins Scratch** en haut à droite de l'écran. Dans la fenêtre qui apparaît (figure A-1), entrez le nom d'utilisateur et le mot de passe, puis cliquez sur **Suivant**.



Figure A-1 : Première fenêtre du processus de création de compte

2. Dans la seconde fenêtre (figure A-2), entrez votre date de naissance, votre sexe ainsi que votre pays, puis cliquez sur **Suivant**.



Figure A-2 : Deuxième fenêtre du processus de création de compte

Copyright © 2016 Eyrolles.

3. Rentrez votre adresse électronique dans la troisième fenêtre (figure A-3), puis cliquez sur **Suivant**.



Figure A-3 : Troisième fenêtre du processus de création de compte

4. Vous verrez ensuite une fenêtre de bienvenue dans la communauté des utilisateurs de Scratch (figure A-4). Cliquez sur **OK Allons-y !** et vous serez connecté à votre nouveau compte.

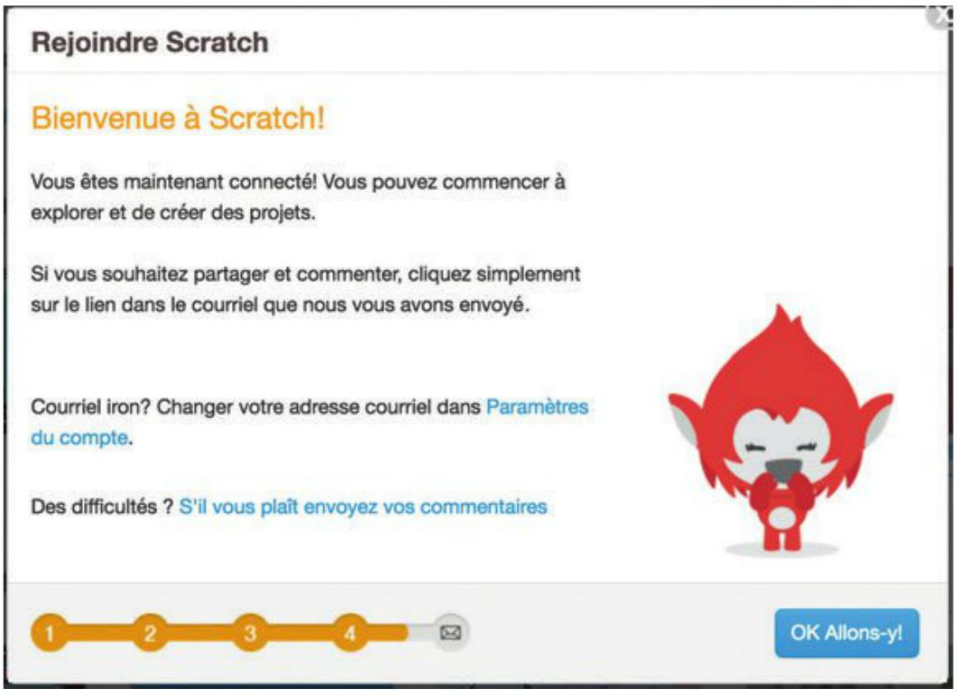


Figure A-4 : Dernière fenêtre du processus de création de compte

Copyright © 2016 Eyrolles.

La barre de navigation en haut de l'écran affiche votre nom d'utilisateur (figure A-5). Utilisez les quatre liens (**Créer, Explorer**, **Discuter** et **Aide**) de la barre de navigation pour démarrer l'éditeur de projets de Scratch, explorer les projets disponibles, collaborer avec d'autres Scratchers et trouver des conseils utiles ainsi que des ressources additionnelles.

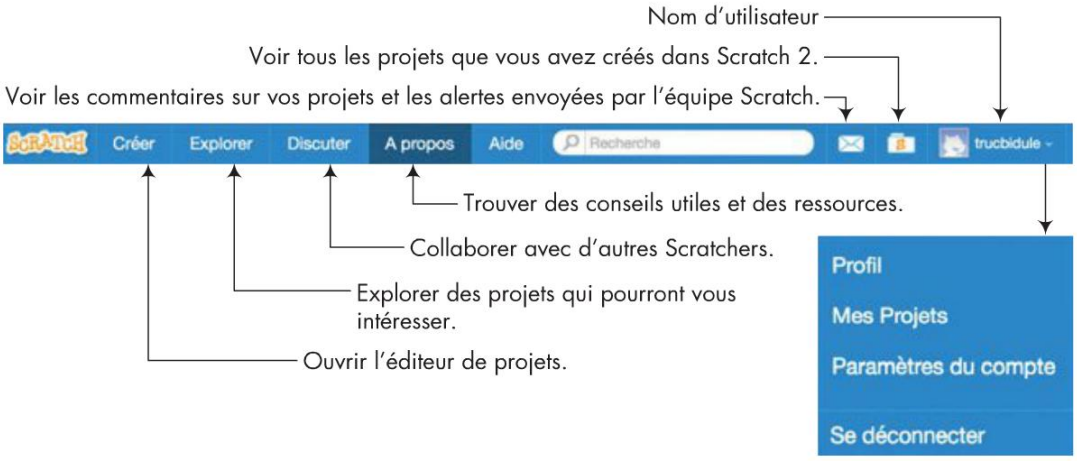


Figure A-5 : La barre de navigation d'un utilisateur connecté

La section suivante présente quelques-unes des fonctions qui seront disponibles lorsque vous serez connecté à votre compte Scratch.

Utilisation du sac à dos

Le sac à dos (disponible uniquement si vous êtes connecté) vous permet de copier des lutins, des scripts, des costumes, des arrière-plans et des sons à partir de n'importe quel projet et de les utiliser dans vos propres projets. Cliquez sur le lien **Explorer** (figure A-5) pour aller à la page d'exploration de projets présentée de la figure A-6. Vous pourrez y tester des projets Scratch créés par d'autres personnes.

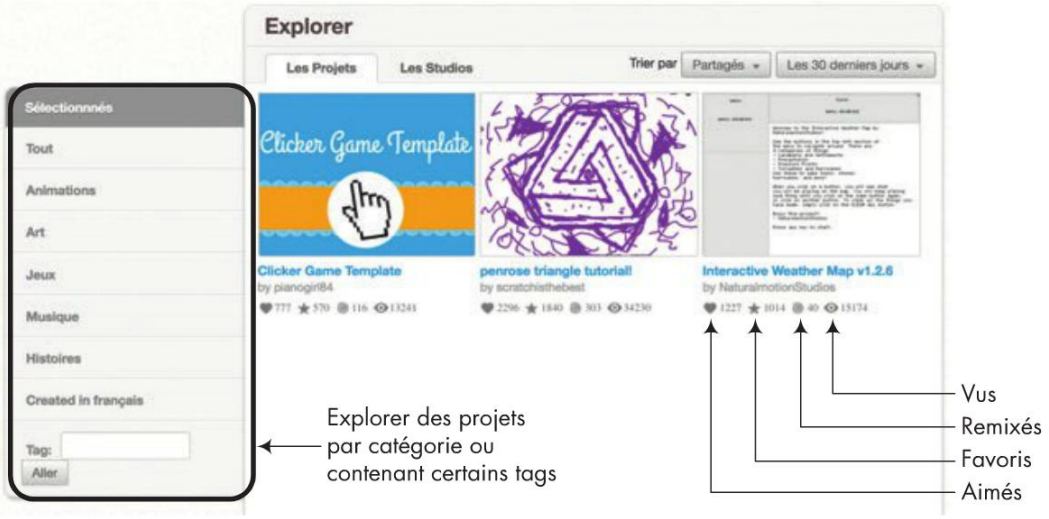


Figure A-6 : Page d'exploration de projets

Copyright © 2016 Eyrolles.

Vous aurez la possibilité de classer les projets par catégorie, rechercher des projets contenant certains mots et les trier selon différents critères (partagés, les plus consultés, les préférés, ou les plus souvent remixés). Lorsque vous trouverez un projet que vous voudrez explorer, double-cliquez sur sa miniature (figure A-7).

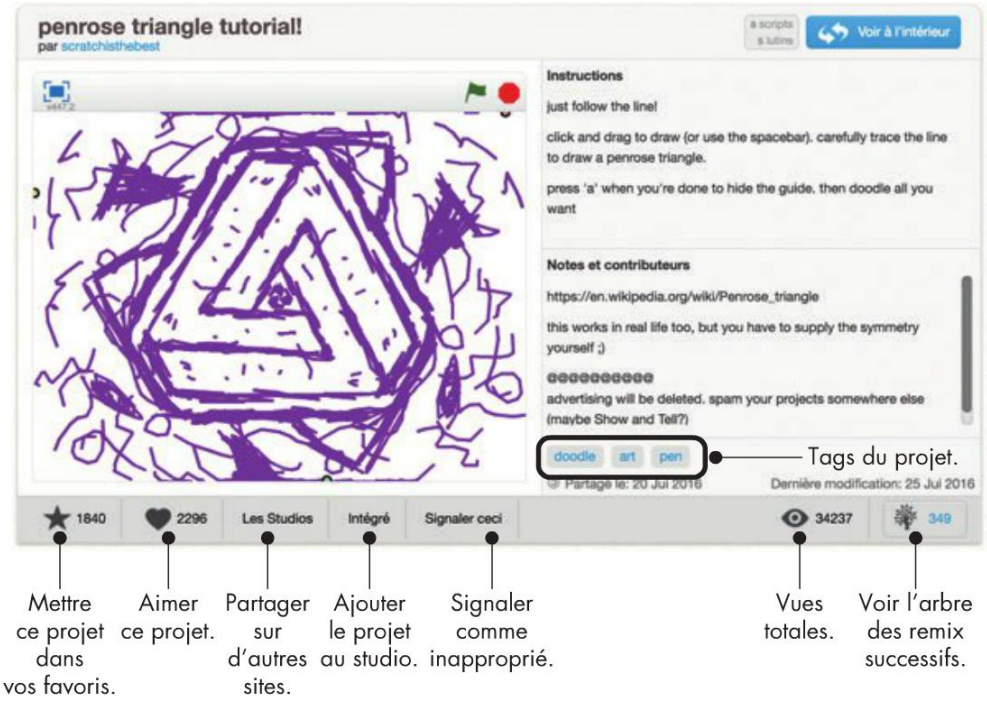


Figure A-7 : Un exemple de page de projet

Cliquez sur le bouton **Voir à l'intérieur** dans le coin supérieur droit de la figure A-7 pour voir le contenu de ce projet (figure A-8).

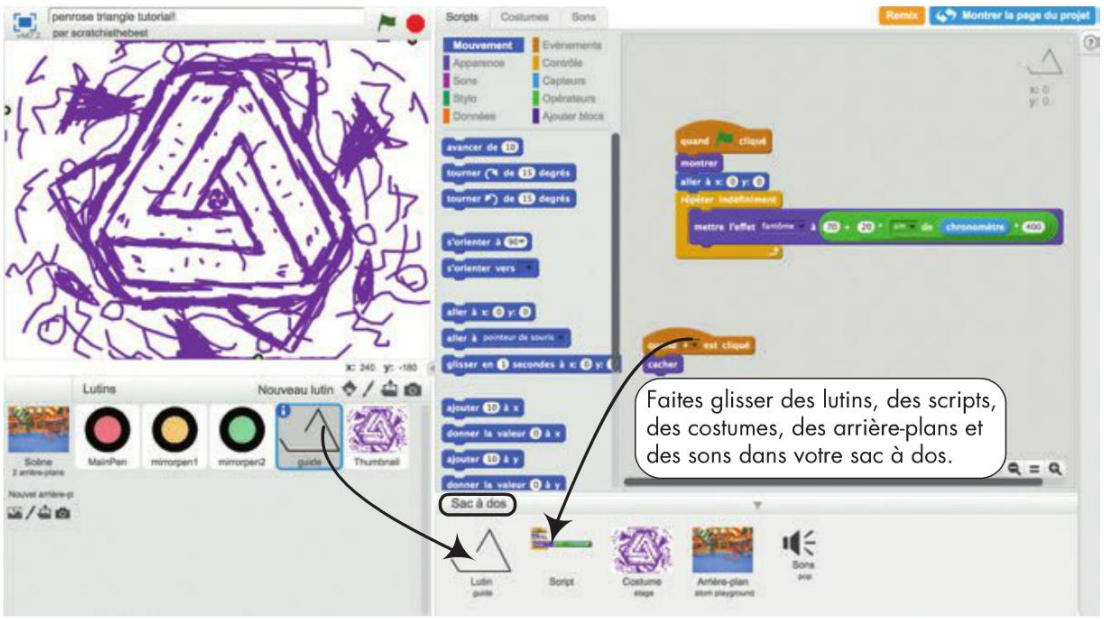


Figure A-8 : Visualisation du contenu d'un projet d'un autre Scratcher

Copyright © 2016 Eyrolles.

Si vous voulez utiliser des parties de ce projet (lutins, scripts, costumes, arrière-plans et sons) dans l'une de vos propres applications, faites simplement glisser ces parties dans votre sac à dos. Pour supprimer un élément du sac à dos, cliquez droit dessus et sélectionnez **Supprimer** dans le menu contextuel.

Le contenu de votre sac à dos est sauvegardé sur un serveur de Scratch, afin que vous ne le perdiez pas en vous déconnectant. Pour utiliser un de ces éléments, faites-le simplement glisser du sac à dos vers votre projet.

Partage de vos propres projets

Il y a de nombreuses manières de commencer à programmer avec Scratch. Vous pouvez créer un nouveau projet, remixer un projet qui a été partagé sur le site de Scratch, ou ouvrir un vieux projet et le modifier. Nous allons explorer chacune de ces options.

Démarrer un nouveau projet

Pour démarrer un nouveau projet, cliquez sur le lien **Créer** dans la barre de navigation afin d'ouvrir l'éditeur de projets de Scratch (figure A-9).

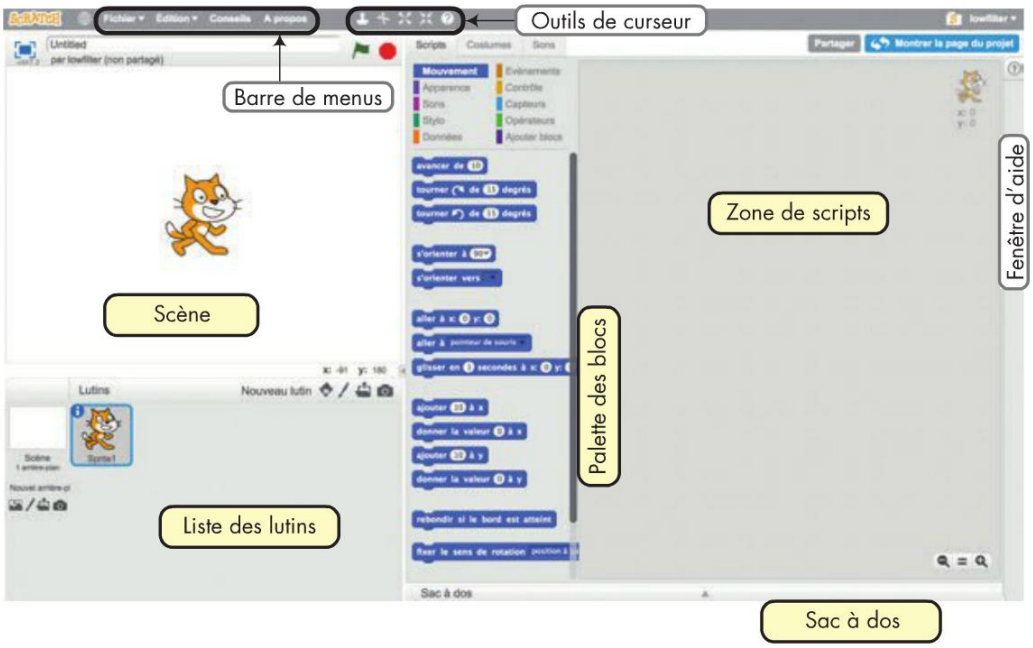


Figure A-9 : L'éditeur de projets de Scratch pour un utilisateur connecté

Cette interface est très similaire à celle que nous avons vue lorsque nous ne sommes pas connectés, mais il existe quelques différences importantes :

- Le volet du sac à dos est visible.
- Deux nouveaux boutons (**Partager** et **Montrer la page du projet**) apparaissent dans le coin supérieur droit.
- L'icône de valise et le nom d'utilisateur apparaissent à droite de la barre d'outils.
- Le menu **Fichier** possède de nouvelles options.

Copyright © 2016 Eyrolles.

La barre d'outils et ses nouvelles options sont présentées plus en détail sur la figure A-10.

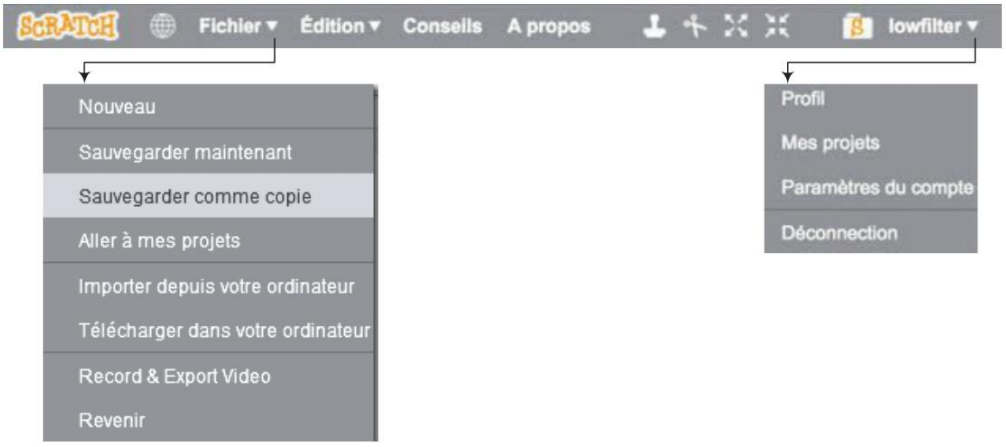


Figure A-10 : La barre d'outils d'un utilisateur connecté

Lorsque vous êtes connecté, Scratch sauvegarde automatiquement votre travail dans le cloud (sur le serveur de Scratch), mais il est toujours judicieux de cliquer sur **Sauvegarder** avant de quitter le site. L'option **Sauvegarder comme copie** conserve votre projet actuel avec un nom différent. Si votre projet actuel s'appelle *Test*, par exemple, le nouveau projet s'appellera *Test copy*. Vous pouvez changer ce nom en en entrant un nouveau dans la zone de texte correspondante. L'option **Revenir** annule tout changement que vous aurez pu faire depuis l'ouverture du projet actuel.

Si vous voulez enregistrer vos projets sur votre ordinateur plutôt que sur le cloud, utilisez l'option **Télécharger dans votre ordinateur**. L'option **Importer depuis votre ordinateur**, quant à elle, vous permet de charger à partir de votre ordinateur vers l'éditeur de projets. Vous pouvez utiliser cette option pour télécharger des projets créés avec Scratch 1.4 et les convertir au format Scratch 2.

Remixer un projet

Cliquez sur le bouton **Remix** lorsque vous avez des idées à ajouter à un projet Scratch. Cela copie le projet sélectionné dans votre compte et vous donne un point de départ pour travailler. Vous pouvez également cliquer sur le bouton **Voir l'arbre des remix successifs** sur la page du projet (figure A-7) pour étudier comment le projet a évolué à travers le temps et choisir la branche à partir de laquelle vous voulez le copier.

Si vous partagez votre projet remixé, la page du projet présentera le nom des créateurs originaux ainsi que des liens vers leurs versions.

La page de projet

Cliquez sur le bouton **Montrer la page du projet** dans le coin supérieur droit de la figure A-9 pour éditer votre page de projet, présentée sur la figure A-11. Vous pouvez entrer des instructions pour les personnes qui utilisent votre application, remercier celles dont les idées vous auraient inspirés ou dont vous auriez utilisé le travail, et spécifier des mots-clés qui aideront les autres utilisateurs à trouver votre application.

Copyright © 2016 Eyrolles.

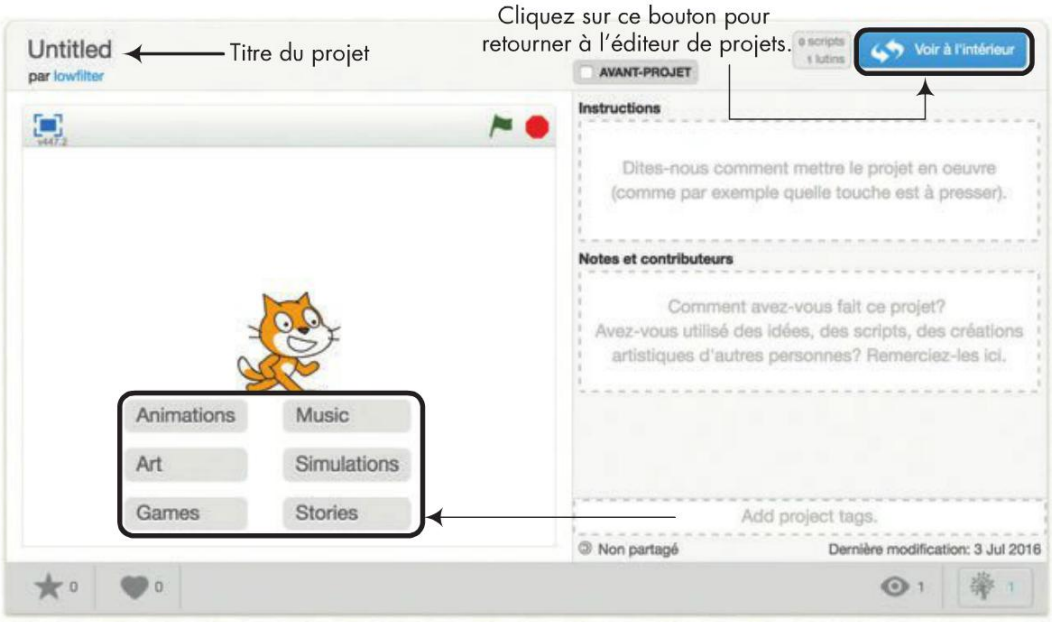


Figure A-11 : Une page de projet

Partager votre projet

Une fois que vous avez terminé votre application, vous pouvez la partager avec la communauté Scratch en cliquant sur le bouton **Partager**. Tout le monde peut alors le trouver en ligne et regarder à l'intérieur.

Pour voir la liste de tous vos projets, cliquez sur votre nom d'utilisateur et sélectionnez **Mes Projets** dans le menu déroulant (vous pouvez également cliquer sur l'icône en forme de valise). Cela vous mènera vers la page Mes Projets (figure A-12).



Figure A-12 : La page Mes projets

La page Mes projets vous permet de contrôler et d'observer différents aspects de vos projets et studios. Vous pouvez créer, partager, éditer, ne plus partager ou supprimer vos projets à partir de cette page. Vous pouvez également créer des *studios*, c'est-à-dire des collections qui servent à regrouper facilement des projets liés.

NOTE *Si vous supprimez l'un de vos projets non partagés, celui-ci sera déplacé dans votre dossier Corbeille, qui se comporte comme une poubelle de recyclage pour projets. L'interface de la Corbeille vous permet de restaurer un projet supprimé de la page Mes projets.*

Copyright © 2016 Eyrolles.

INDEX

Symboles

+ (addition), 21
/ (division), 21
* (multiplication), 21
- (soustraction), 21

A

addition, 21
aléatoire, 21
AnatomieFleur.sb2, 242
animation, 48
Animation.sb2, 48
apparence, 48
AppliCarre.sb2, 69
Aquarium.sb2, 64
Argent_PasDeCode.sb2, 36
argument, 80
arrêt, 160
arrière-plan, 6, 11
arrondi, 21
AttrapePommes_PasDeCode.sb2, 39

B

barre d'outils, 12
BeatsDemo.sb2, 54
BinaireDécimal.sb2, 199
Blague.sb2, 65
bloc, 20
    arguments, 7
    personnalisé, 75
        paramètre d'entrée, 77
        procédures imbriquées, 81
    boucle, 156
    catégories, 6
    chapeaux, 20
    commande, 20
    connexion, 7
    contrôle, 20
    copier, 9
    déclenchement, 20
    déplacer, 8
    empilement, 20
    entrées, 7
    fonction, 20
    forme, 92
    moniteur, 21
    onglet, 6
    palettes, 6
    rapporteurs, 20
Boole, George, 125
booléen
    algèbre, 125
    bloc de fonction, 93
    condition, 125
    expression, 126
boucle
    bloc, 156
    comptée, 156
    conditionnelle, 156
    définie, 156
    fin, 162
    imbriquer, 167
    indéfinie, 156
    infinie, 156
        conditionnelle, 158
        inconditionnelle, 158
BouclesImbriquées1.sb2, 167
BouclesImbriquées2.sb2, 168

C

CalculateurdAire.sb2, 132
CarresPivotes.sb2, 82
CarrésRotatifs.sb2, 34
CentreDeRotation.sb2, 14
chaîne de caractères
    comparaison, 126, 188
    comptage du nombre de caractères spéciaux, 186
    manipulation, 189
    traitement, 185
ChangeDeChapeau.sb2, 64
ChasseLaBalle.sb2, 30
ChasseurdOiseau.sb2, 173
ChatQuiEternue.sb2, 51
ChuteLibre.sb2, 177

Copyright © 2016 Eyrolles.

CircuitSérie.sb2, 110
ClassificationDeTriangles.sb2, 142
ClassifQuad.sb2, 236
ClicLeVisage.sb2, 49
clone, 42
CloneIDs.sb2, 105
ClonesEtVarGlobales.sb2, 106
commande
    stop, 160
    stylo, 31
comparaison, 124
    chaîne de caractères, 126
    nombres décimaux, 139
ComparaisonDeFractions.sb2, 214
ComptagedEléments.sb2, 229
ComptageDeVoyelles.sb2, 186
compte
    création, 247
    sac à dos, 250
CompterAvecMontantConst.sb2, 165
compteur, 164
coordonnées
    souris, 4
CorrigeMonOrthographe.sb2, 191, 193
costume, 9
CouleurDeScène.sb2, 108

D

Dames.sb2, 82
DanseSurScene.sb2, 57
décision, 123
    structure, 128
DemanderEtAttendre.sb2, 118
DemanderEtAttendre2.sb2, 119
DemanderEtAttendre3.sb2, 119
dessin, 31
DessinCarré.sb2, 33
DessinFacile.sb2, 31
DevineMesCoordonnées.sb2, 140
DevineMonNuméro.sb2, 151
division, 21
donnée
    entrée par l'utilisateur, 118
    type, 92
drapeau, 129
    vert, 4

E

éditeur d'images, 13
effet graphique, 50
EffetsGraphiques.sb2, 50
EntréesUtilisateur.sb2, 118
ÉquationdUneDroite.sb2, 147
estampiller, 35
et, 135

F

FeudArtifice_PasDeCode.sb2, 60
FeuRouge.sb2, 49
Fleurs.sb2, 70
Fleurs2.sb2, 73
FloconDeFleur.sb2, 87
fonction mathématique, 22
FrèreJacques.sb2, 55

H

Histogramme.sb2, 223
HorlogeAnalogique.sb2, 171

I

image, 13
    centre, 13
indice, 216

J

JeuDeTri.sb2, 242

L

langage de programmation, 2
liste, 215
    accéder à un élément, 220
    ajouter un élément, 219
    commande, 218
    comptage d'éléments, 229
    création, 216
    des lutins, 5
    indice, 216
    insérer un élément, 219
    localisation d'élément, 228
    maximum, 226
    médiane, 233
    minimum, 226
    moniteur, 217
    moyenne, 227
    numérique, 226
    remplacer un élément, 219
    remplir, 222
    sentinelle, 222
    supprimer un élément, 219
    tri, 230

Copyright © 2016 Eyrolles.

Localisation.sb2, 228
LoidOhm.sb2, 109
Louchébem.sb2, 190-191
Loup.sb2, 63
lutin, 2
  ajout, 5
  cacher/montrer, 5
  clone, 42
  costume, 9
  dupliquer, 5
  exporter, 5
  importer, 5
  informations, 11
  miniature, 5
  parler, 50
  penser, 50
  plan, 52
  sons, 10
  supprimer, 5
  taille, 51
  visibilité, 51

M

MagicienMathématique.sb2, 238
Maison.sb2, 85
médiane, 233
Médiane.sb2, 233
menu, 132
message, 68
  coordination de plusieurs lutins, 70
  créer une procédure, 73
  émission, 69
  réception, 69
miniature
  lutin, 5
  scène, 6
modulo, 21
MoléculesEnMouvement.sb2, 182
moniteur, 21, 107
  afficher, 107
  apparence, 107
  liste, 217
  portée, 108
  potentiomètre, 108
Mots.sb2, 64
Moulin.sb2, 35
mouvement
  absolu, 26
  commande, 25
  relatif, 27
multiplication, 21
musique, 55

N

Nature.sb2, 66
nombre aléatoire, 21

O

onglet
  Arrière-plans, 11
  Costumes, 9
  Scripts, 6
  Sons, 10
opérateur
  addition, 21
  arithmétique, 21
  arrondi, 21
  comparaison, 124
  division, 21
  et, 135
  logique, 134
  modulo, 21
  multiplication, 21
  nombre aléatoire, 21
  non, 136
  ou, 135
  soustraction, 21
Orbite.sb2, 182
ou, 135
outils, 12

P

palette
  apparence, 48
  Sons, 53
Palindrome.sb2, 188
panneau stop, 4
paramètre, 80
Pendu.sb2, 202
penser, 50
percussions, 54
PierreFeuilleCiseaux.sb2, 151
plan, 52
Plans.sb2, 53
Poète.sb2, 235-236
Polygone.sb2, 34
pong, 15
Pong.sb2, 15
Pong_PasDeCode.sb2, 15
PortéeDémo.sb2, 100
présentation, 4
PressionSouslEau_PasDeSolution.sb2, 90

Copyright © 2016 Eyrolles.

procédure, 67
    imbriquée, 81
    processus
        descendant, 84
        montant, 87
processus
    descendant, 84
    montant, 87
ProfesseurDeFractions.sb2, 207
programmation
    à base de texte, 2
    visuelle, 2
programme informatique, 2
Projectile.sb2, 179
projet
    nouveau, 252
    page, 253
    par nom
        Attraper des pommes, 39
        Chasseur d'oiseaux, 173
        Classification de quadrilatères, 236
        Classification de triangles, 142
        Convertisseur binaire-décimal, 199
        Danser sur scène, 57
        Devine mes coordonnées, 140
        Équation d'une droite, 147
        Feu d'artifice, 60
        Horloge analogique, 171
        Le poète, 235
        Magicien des mathématiques, 238
        Pendu, 202
        Pong, 15
        Prenez l'argent, 36
        Professeur de fractions, 207
        Quizz sur l'anatomie d'une fleur, 241
        Simulation de chute libre, 177
        Simulation du mouvement d'un projectile, 179
        Suiveur de ligne, 145
        Tir, 196
    partage, 254
    remixer, 253

R

récursivité, 169
RécursivitéLame.sb2, 170
Récursivité.sb2, 169
Réordonne.sb2, 193
répétition, 33, 155
RoseNPétales.sb2, 114

S

sac à dos, 4, 250
scène, 4
    arrière-plan, 6, 11
    miniature, 6
Scratch, 2
    site Internet, 3
script, 2
    zone, 7
sentinelle, 222
si, 128
si et si/sinon imbriqués, 132
si/sinon, 130
SimulateurDeDés_PasDeCode.sb2, 97
son, 10, 53
    fichier, 53
    musique, 55
    percussions, 54
    tempo, 56
    volume, 55
soustraction, 21
Sphère.sb2, 112
stop, 160
StopDemo.sb2, 160
stylo, 31
SuiveurDeLigne.sb2, 145

T

taille, 51
Tape-Taupe.sb2, 122
tempo, 56
Tir.sb2, 196
ToiledAraignée.sb2, 103
Tournesol.sb2, 116
transparent, 14
TriÀBulles.sb2, 230
TrouveLeMontant.sb2, 182
TrouverLaMoyenne.sb2, 227
TrouverMaximum.sb2, 226
type de donnée, 92
    conversion, 93
    forme d'un bloc, 92

V

valider les entrées, 162
variable, 91
    création, 97

Copyright © 2016 Eyrolles.

dans les clones, 105
drapeau, 129
introduction, 94
modification, 102
moniteur, 107
nom, 98
portée, 100
type de donnée, 92
VérificationDeMotDePasse.sb2, 164
visibilité, 51

Z

Zebre.sb2, 63
zone de scripts, 7

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

À PROPOS DE L'AUTEUR ET DU VALIDATEUR TECHNIQUE

Docteur en génie électrique de l'université de Wayne State (Michigan), Majed Marji est également titulaire d'un MBA en stratégie d'entreprise de l'université de Davenport (Michigan). Il possède plus de 15 ans d'expérience dans le secteur de la construction automobile, au sein duquel il a développé beaucoup d'applications informatiques pour la chaîne d'acquisition temps-réel, le contrôle des dispositifs, la gestion du banc d'essai, l'analyse technique de données, les contrôleurs intégrés, la télématique, les véhicules hybrides et les groupes motopropulseurs. Dr. Marji est par ailleurs professeur auxiliaire du département Génie électrique de l'université de Wayne State. Il y donne entre autres des cours sur les systèmes de communication, la vision industrielle, les microprocesseurs, les systèmes de contrôle, les algorithmes et les structures de données.

Tyler Watts enseigne l'informatique au collège à Kansas City (Kansas) et à l'université du Missouri-Kansas City. Il utilise Scratch depuis 2009 pour combattre la fracture numérique et apprendre à ses élèves à penser comme des informaticiens. Il considère que la programmation est un moyen personnel d'expression, qu'il faut appréhender comme s'il s'agissait d'une discipline artistique.

Copyright © 2016 Eyrolles.

APPRENEZ À CODER À VOS ÉLÈVES DÈS 5 ANS !

NOUVELLE MATIÈRE DU PROGRAMME, DU CE1 À LA 3E !

SCRATCH



Dès 5 ans
160 pages – 15,90 €



Dès 8 ans
160 pages – 15,90 €

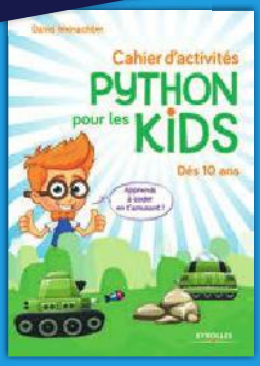


Dès 8 ans
64 pages – 12 €



Dès 10 ans
288 pages – 25 €

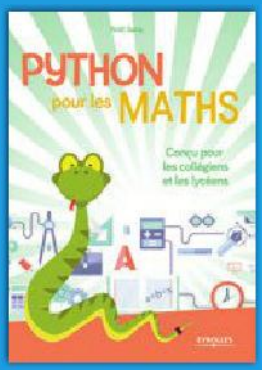
PYTHON



Dès 10 ans
64 pages – 12 €



Dès 10 ans
332 pages – 22,90 €

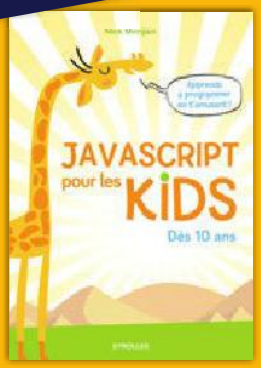


Dès 10 ans
300 pages – 22,90 €



Dès 10 ans
350 pages – 24,90 €

WEB



Dès 10 ans
364 pages – 22,90 €
À paraître en 2016



Dès 10 ans
152 pages – 19,90 €
À paraître en 2016

MAKERS



Dès 10 ans
352 pages – 25 €



Dès 8 ans
64 pages – 12 €
À paraître en 2016

EYROLLES

Pour en savoir plus : www.editions-eyrolles.com/go/coderaveceyrolles