

Comprendre l'architecture d'une application

Ce chapitre examine la structure d'une application du point de vue d'un programmeur. Cela commence par l'analogie traditionnelle selon laquelle une application est comme une recette, puis reconceptualise une application comme un ensemble de composants qui répondent aux événements. Le chapitre examine également comment les applications peuvent poser des questions, répéter, mémoriser et communiquer avec le Web, tout cela étant décrit plus en détail dans les chapitres suivants.

De nombreuses personnes peuvent vous dire d'où provient une application du point de vue d'un utilisateur, mais comprendre de quoi il s'agit du point de vue d'un programmeur est plus compliqué. Les applications ont une structure interne que vous devez comprendre afin de les créer efficacement.

Une façon de décrire les composants internes d'une application consiste à la diviser en deux parties : ses composants et ses comportements. En gros, celles-ci correspondent aux deux fenêtres principales que vous utilisez dans App Inventor : vous utilisez le concepteur de composants pour spécifier les objets (composants) de l'application et vous utilisez l'éditeur de blocs pour programmer la façon dont l'application répond à l'utilisateur et aux événements externes (le comportement de l'application).

La figure 14-1 donne un aperçu de cette architecture d'application. Dans ce chapitre, nous allons explorer cette architecture en détail.

Figure 14-1.



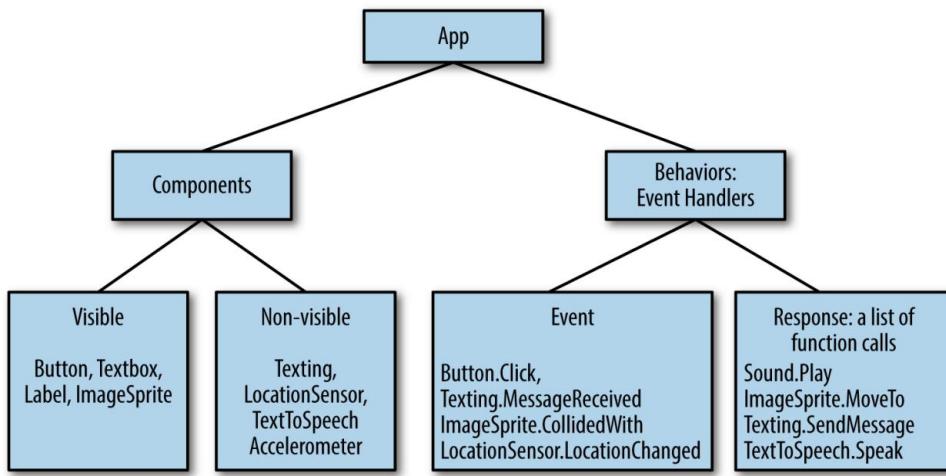


Figure 14-2. L'architecture interne d'une application App Inventor

Composants

Il existe deux principaux types de composants dans une application : visibles et non visibles. Les composants visibles de l'application sont ceux que vous pouvez voir au lancement de l'application : boutons, zones de texte et étiquettes. Celles-ci sont souvent appelées interface utilisateur de l'application.

Les composants non visibles sont ceux que vous ne pouvez pas voir, ils ne font donc pas partie de l'utilisateur. interface. Au lieu de cela, ils donnent accès aux fonctionnalités intégrées de l'appareil ; par exemple, le composant Texting envoie et traite des SMS, le composant LocationSensor détermine l'emplacement de l'appareil et le composant TextToSpeech parle.

Les composants non visibles sont la technologie présente dans l'appareil : de petites abeilles ouvrières qui effectuent des tâches pour votre application.

Les composants visibles et non visibles sont définis par un ensemble de propriétés. Les propriétés sont des emplacements mémoire permettant de stocker des informations sur le composant. Les composants visibles tels que les boutons et les étiquettes ont des propriétés telles que la largeur, la hauteur et l'alignement, qui définissent ensemble l'apparence du composant.

Les propriétés des composants sont comme les cellules d'une feuille de calcul : vous les modifiez dans le Concepteur de composants pour définir l'apparence initiale d'un composant. Vous pouvez également modifier les valeurs avec des blocs.

Comportement

Les composants de l'application sont généralement simples et faciles à comprendre : une zone de texte permet de saisir des informations, un bouton permet de cliquer, etc. Le comportement d'une application, en revanche, est conceptuellement difficile et souvent complexe. Le comportement définit comment le

L'application doit répondre aux événements, à la fois initiés par l'utilisateur (par exemple, un clic sur un bouton) et externes (par exemple, un SMS arrivant sur le téléphone). La difficulté de spécifier un tel comportement interactif est la raison pour laquelle la programmation est si difficile.

Heureusement, App Inventor fournit un langage de haut niveau basé sur des blocs pour spécifier les comportements. Les blocs font que les comportements de programmation ressemblent davantage à l'assemblage de pièces de puzzle, par opposition aux langages de programmation traditionnels basés sur du texte, qui impliquent l'apprentissage et la saisie de grandes quantités de code. Et App Inventor est conçu pour rendre la spécification de comportements de réponse aux événements particulièrement simple. Les sections suivantes fournissent un modèle permettant de comprendre le comportement des applications et de le spécifier dans App Inventor.

Une application comme recette

Traditionnellement, les logiciels sont souvent comparés à une recette. Comme une recette, une application traditionnelle suit une séquence linéaire d'instructions que l'ordinateur doit exécuter, comme illustré dans la figure 14-2.

Une application typique peut démarrer une transaction bancaire (A), effectuer certains calculs et modifier le compte d'un client (B), puis imprimer le nouveau solde sur l'écran (C).

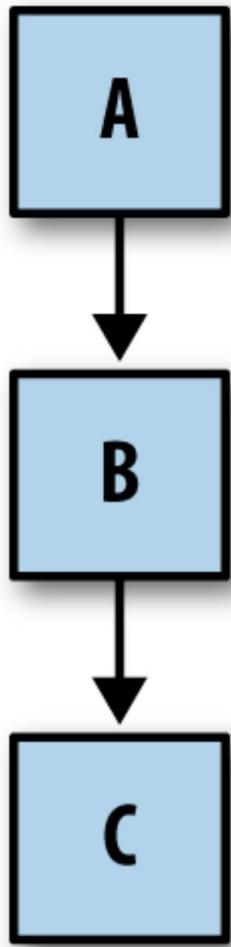


Figure 14-3. Les logiciels traditionnels suivent une séquence linéaire d'instructions

Une application en tant qu'ensemble de gestionnaires d'événements

L'application en tant que paradigme de recette s'adapte bien aux premiers ordinateurs de calcul, mais elle n'est pas idéale pour les téléphones mobiles, le Web et, en général, la plupart des ordinateurs utilisés aujourd'hui. La plupart des logiciels modernes n'exécutent pas un ensemble d'instructions dans un ordre prédéterminé ; au lieu de cela, il réagit aux événements, le plus souvent aux événements initiés par l'utilisateur final de l'application. Par exemple, si l'utilisateur appuie sur un bouton, l'application répond en effectuant une opération (par exemple, l'envoi d'un message texte). Pour les téléphones et appareils à écran tactile, le fait de faire glisser votre doigt sur l'écran est un autre événement. Le

L'application peut répondre à cet événement en traçant une ligne à partir du point où votre doigt touche pour la première fois l'écran jusqu'au point où vous l'avez soulevé.

Les applications modernes sont mieux conceptualisées comme des machines à réponse aux événements. Les applications font incluent des recettes (des séquences d'instructions), mais chaque recette n'est exécutée qu'en réponse à un événement, comme le montre la figure 14-3.

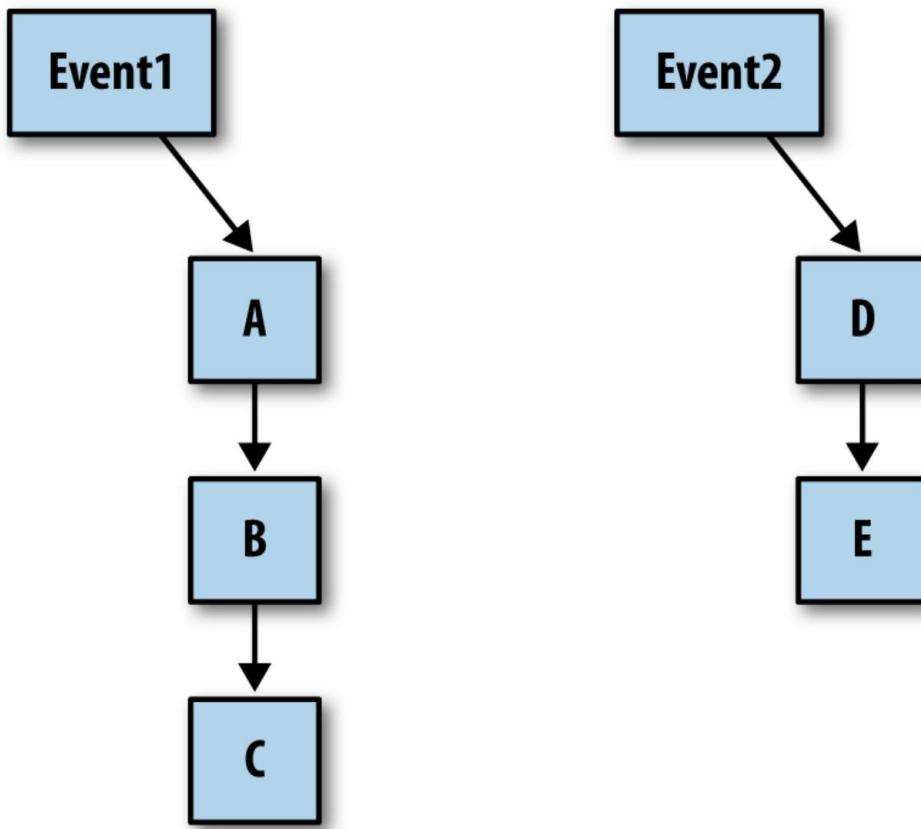


Figure 14-4. Une application comme plusieurs recettes accrochées aux événements

Lorsque des événements se produisent, l'application réagit en appelant une séquence de fonctions. Les fonctions sont des choses que vous pouvez faire sur ou avec un composant ; il peut s'agir d'opérations telles que l'envoi d'un texte SMS, ou d'opérations de modification de propriétés telles que la modification du texte dans une étiquette de l'interface utilisateur. Appeler ou invoquer une fonction signifie exécuter la fonction, la réaliser. Nous appelons un événement et l'ensemble des fonctions exécutées en réponse à celui-ci un gestionnaire d'événements.

De nombreux événements sont initiés par l'utilisateur final, mais certains ne le sont pas. Une application peut réagir aux événements qui se produisent à l'intérieur du téléphone, tels que les modifications apportées à son capteur d'orientation et à l'horloge (c'est-à-dire le passage du temps), ou elle peut répondre à des événements provenant de l'extérieur.

le téléphone, comme un SMS ou un appel entrant provenant d'un autre téléphone, ou des données provenant du Web (voir Figure 14-4).

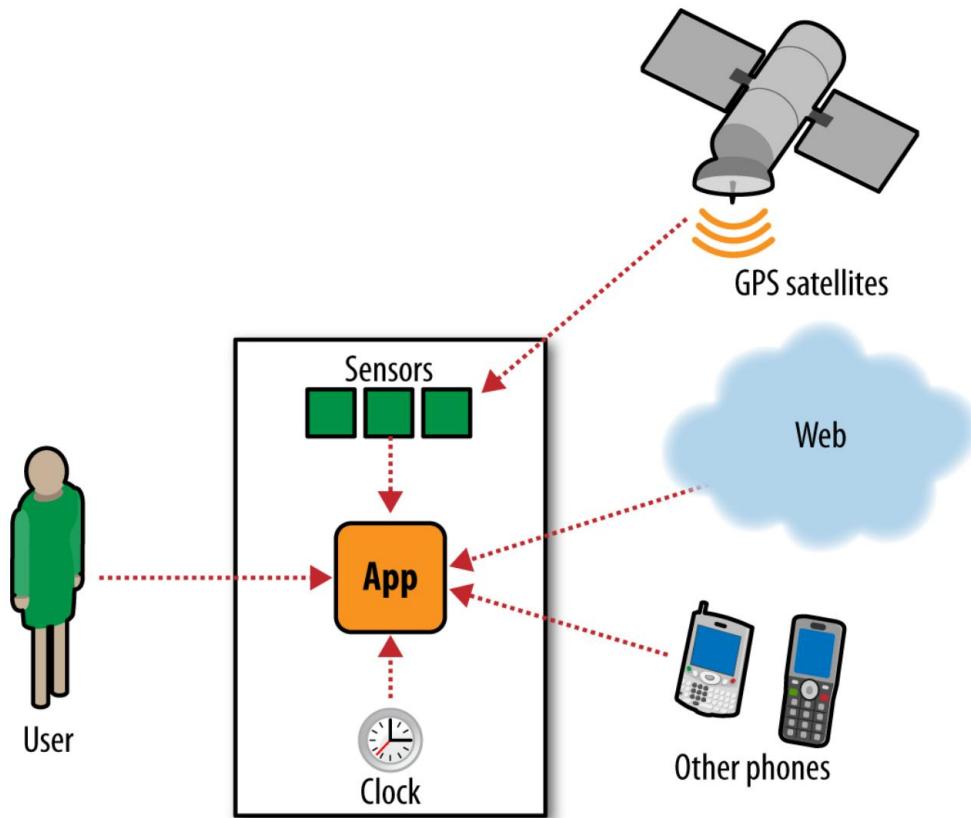


Figure 14-5. Une application peut répondre à des événements internes et externes

L'une des raisons pour lesquelles la programmation App Inventor est si intuitive est qu'elle est basée directement sur ce paradigme événement-réponse ; les gestionnaires d'événements sont des primitives dans le langage (dans de nombreux langages, ce n'est pas le cas). Vous commencez à définir un comportement en faisant glisser un bloc d'événement, qui a la forme « Quand <événement> fait ». Par exemple, considérons une application, SpeakIt, qui répond aux clics sur un bouton en prononçant à haute voix le texte que l'utilisateur a tapé dans une zone de texte. Cette application pourrait être programmée avec un seul gestionnaire d'événements, comme le montre la figure 14-5.



Figure 14-6. Un gestionnaire d'événements pour une application SpeakIt

Ces blocs spécifient que lorsque l'utilisateur clique sur le bouton nommé SpeakItButton, le composant TextToSpeech doit prononcer les mots que l'utilisateur a tapés dans la zone de texte nommée TextBox1. La réponse est l'appel à la fonction TextToSpeech1.Speak. L'événement est SpeakItButton.Click. Le gestionnaire d'événements inclut tous les blocs de la figure 14-5.

Avec App Inventor, toutes les activités se produisent en réponse à un événement. Votre application ne devrait pas contenir des blocs en dehors du moment où faire un bloc d'un événement. Par exemple, les blocs de la figure 14-6 n'apportent rien lorsque vous naviguez seul.

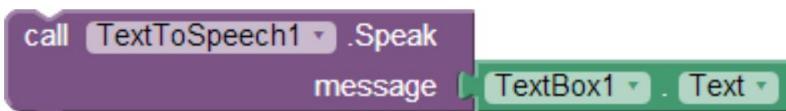


Figure 14-7. Les blocs flottants ne feront rien en dehors d'un gestionnaire d'événements

Types d'événements

Les événements pouvant déclencher une activité appartiennent aux catégories répertoriées dans le tableau 14-1.

Tableau 14-1. Événements pouvant déclencher une activité

Type d'événement	Exemple
Événement initié par l'utilisateur	Lorsque l'utilisateur clique sur le bouton 1, faites...
Événement d'initialisation	Lorsque l'application est lancée, faites...
Événement de minuterie	Lorsque 20 millisecondes s'écoulent, faites...
Événement d'animation	Lorsque deux objets entrent en collision, faites...
Événement externe	Lorsque le téléphone reçoit un SMS, faites...

ÉVÉNEMENTS INITIÉS PAR L'UTILISATEUR

Les événements lancés par l'utilisateur constituent le type d'événement le plus courant. Avec les formulaires de saisie, c'est généralement l'utilisateur qui appuie sur un bouton qui déclenche une réponse de l'application. Des applications plus graphiques répondent aux touches et aux glissements.

ÉVÉNEMENTS D'INITIALISATION

Parfois, votre application doit exécuter certaines fonctions immédiatement après le démarrage, et non en réponse à une activité de l'utilisateur final ou à un autre événement. Comment cela s'intègre-t-il dans le paradigme de la gestion des événements ?

Les langages de gestion d'événements tels qu'App Inventor considèrent le lancement de l'application comme un événement. Si vous souhaitez que des fonctions spécifiques soient exécutées à l'ouverture de l'application, faites glisser un bloc d'événement Screen1.Initialize et placez-y les blocs d'appel de fonction pertinents.

Par exemple, dans le jeu MoleMash (Chapitre 3), la procédure MoveMole est appelée au démarrage de l'application (voir Figure 14-7) pour placer la taupe de manière aléatoire.

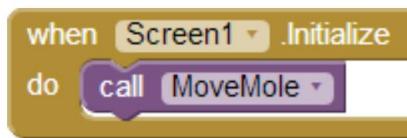


Figure 14-8. Utilisation d'un bloc d'événement Screen1.Initialize pour déplacer la taupe au lancement de l'application

ÉVÉNEMENTS MINUTERIE

Certaines activités dans une application sont déclenchées par le passage du temps. Vous pouvez considérer une animation comme un objet qui se déplace lorsqu'il est déclenché par un événement de minuterie. App Inventor dispose d'un composant Clock que vous pouvez utiliser pour déclencher des événements de minuterie. Par exemple, si vous vouliez qu'une balle sur l'écran se déplace de 10 pixels horizontalement à un intervalle de temps défini, vos blocs ressembleraient à la figure 14-8.

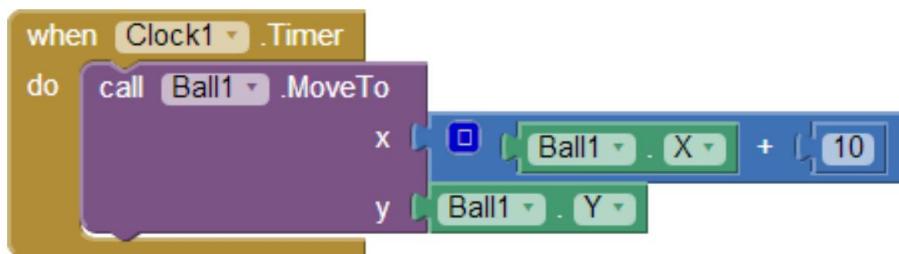


Figure 14-9. Utiliser un bloc d'événements de minuterie pour déplacer une balle à chaque fois que Clock1.Timer se déclenche

ÉVÉNEMENTS D'ANIMATION

Les activités impliquant des objets graphiques (sprites) dans les canevas déclencheront des événements. Vous pouvez ainsi programmer des jeux et autres animations interactives en spécifiant ce qui doit se produire lors d'événements tels qu'un objet atteignant le bord du canevas ou deux objets entrant en collision, comme le montre la figure 14-9. Pour plus d'informations, voir le chapitre 17.



Figure 14-10. Lorsque le sprite FlyingSaucer touche un autre objet, jouez un son

ÉVÉNEMENTS EXTERNES

Lorsque votre téléphone reçoit des informations de localisation des satellites GPS, un événement est déclenché.

De même, lorsque votre téléphone reçoit un SMS, un événement est déclenché (Figure 14-10).

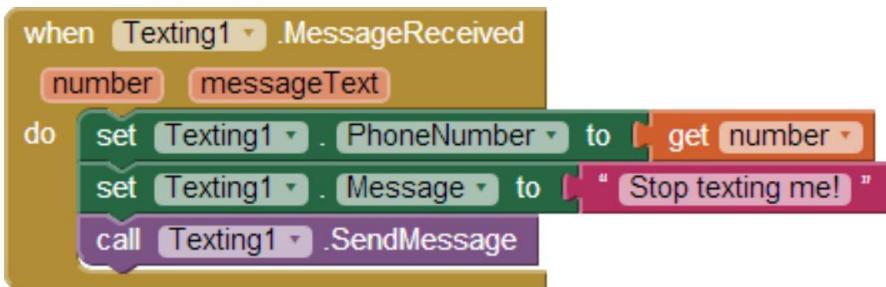


Figure 14-11. L'événement Texting1.MessageReceived est déclenché chaque fois qu'un texte est reçu

De telles entrées externes à l'appareil sont considérées comme des événements, pas différents du fait que l'utilisateur clique sur un bouton.

Ainsi, chaque application que vous créez sera un ensemble de gestionnaires d'événements : un pour initialiser les choses, certains pour répondre aux entrées de l'utilisateur final, certains déclenchés par le temps et certains déclenchés par des événements externes. Votre travail consiste à conceptualiser votre application de cette manière, puis à concevoir la réponse à chaque gestionnaire d'événements.

Les gestionnaires d'événements peuvent poser des questions

Les réponses aux événements ne sont pas toujours des recettes linéaires ; ils peuvent poser des questions et répéter des opérations. « Poser des questions » signifie interroger les données stockées par l'application et déterminer son parcours (branche) en fonction des réponses. Nous disons que ces applications ont des branches conditionnelles. La figure 14-11 illustre une telle branche.

Dans le diagramme, lorsque l'événement se produit, l'application effectue l'opération A puis vérifie une condition. La fonction B1 est exécutée si la condition est vraie. Si la condition est

false, l'application exécute à la place B2. Dans les deux cas, l'application continue d'exécuter la fonction C.

Les tests conditionnels sont des questions telles que « Le score a-t-il atteint 100 ? » ou "Est-ce que le texte que je viens de recevoir vient de Joe ? Les tests peuvent également être des formules plus complexes comprenant plusieurs opérateurs relationnels (inférieur à, supérieur à, égal à) et des opérateurs logiques (et, ou, non).

Vous spécifiez des comportements conditionnels dans App Inventor à l'aide des blocs if et if else . Par exemple, le bloc de la figure 14-12 indiquerait « Vous gagnez ! » si le joueur a marqué 100 points.

Les blocs conditionnels sont abordés en détail au chapitre 18.

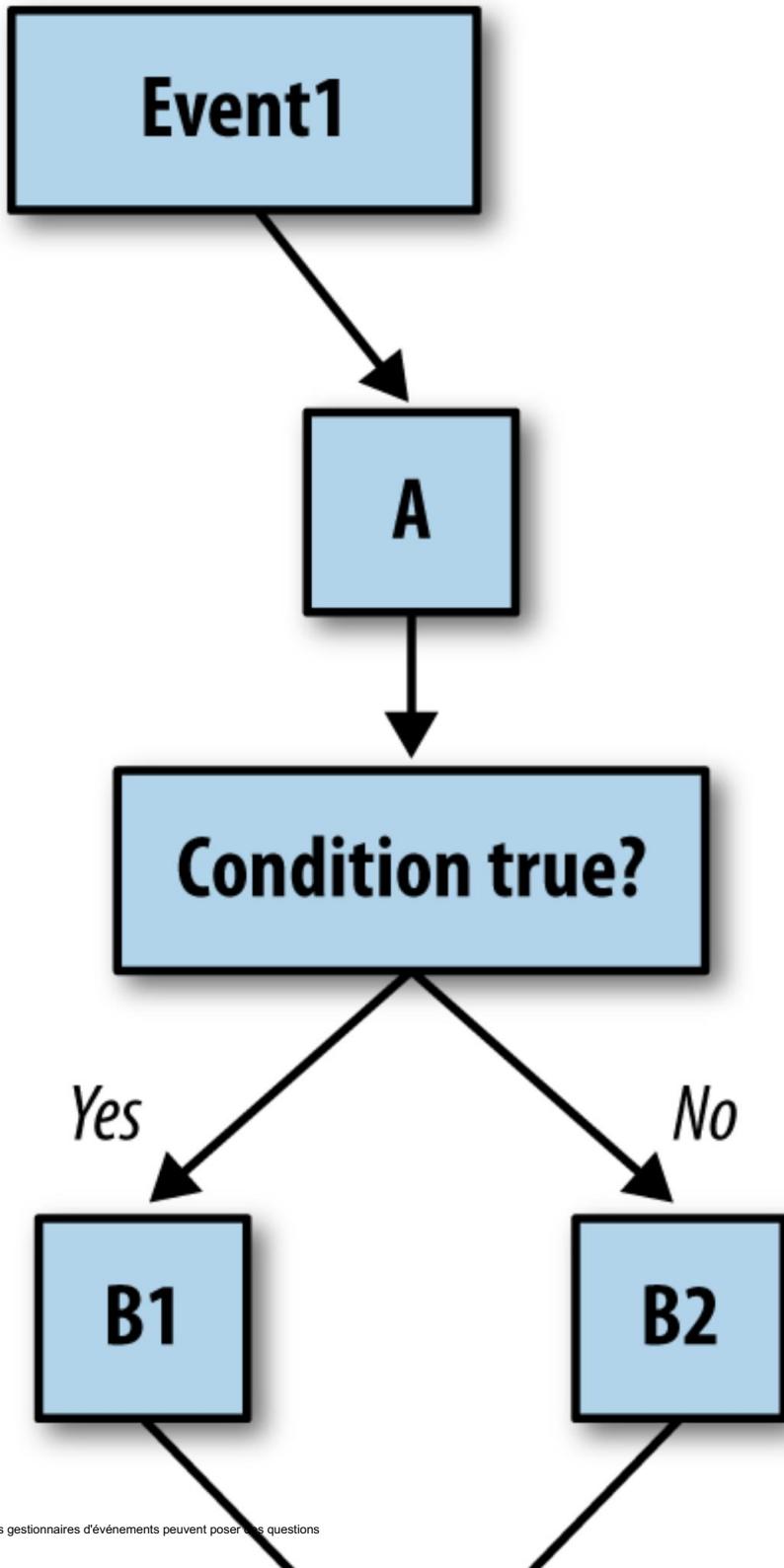




Figure 14-13. Utiliser un bloc if pour signaler une victoire lorsque le joueur atteint 100 points

Les gestionnaires d'événements peuvent répéter des blocs

En plus de poser des questions et de créer des branchements en fonction de la réponse, une réponse à un événement peut également répéter les opérations plusieurs fois. App Inventor fournit un certain nombre de blocs à répéter, notamment le for each et le while do. Les deux renferment d'autres blocs. Tous les blocs de chacun sont exécutés une fois pour chaque élément d'une liste. Par exemple, si vous souhaitez envoyer le même message à une liste de numéros de téléphone, vous pouvez utiliser les blocs de la figure 14-13.

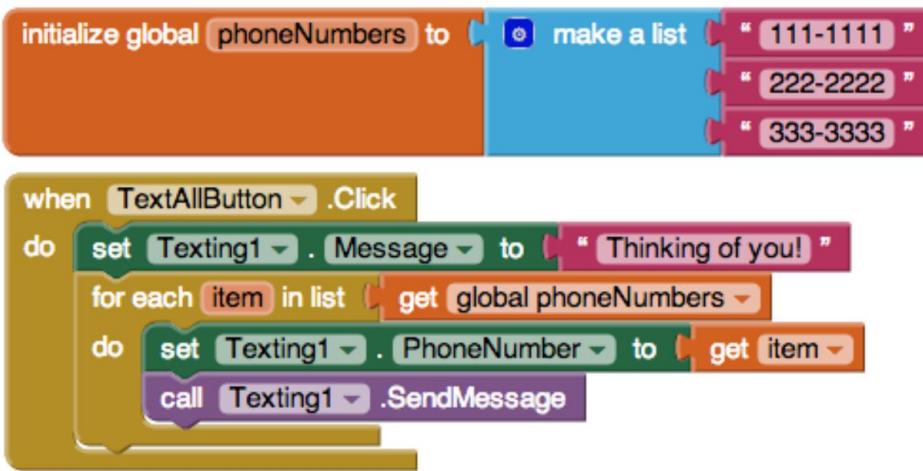


Figure 14-14. Les blocs à l'intérieur de chaque bloc sont répétés pour chaque élément de la liste PhoneNumbers

Les blocs à l'intérieur de chaque bloc sont répétés, dans ce cas, trois fois, car la liste PhoneNumbers contient trois éléments. Dans cet exemple, le message « Je pense à toi... » est envoyé aux trois numéros. Les blocs répétitifs sont abordés en détail au chapitre 20.

Les gestionnaires d'événements peuvent se souvenir de choses

Étant donné qu'un gestionnaire d'événements exécute des blocs, il doit souvent conserver une trace des informations. Les informations peuvent être stockées dans des emplacements mémoire appelés variables, que vous définissez dans l'éditeur de blocs. Les variables sont comme les propriétés des composants, mais elles ne sont associées à aucun composant particulier. Dans une application de jeu, par exemple, vous pouvez définir une variable appelée score, et vos gestionnaires d'événements modifieront sa valeur lorsque l'utilisateur fera quelque chose en conséquence. Les variables stockent temporairement les données pendant l'exécution d'une application ; lorsque vous fermez l'application, les données sont perdues et ne sont plus disponibles.

Parfois, votre application doit mémoriser des éléments non seulement pendant son exécution, mais aussi lorsqu'elle est fermée puis rouverte. Si vous avez suivi un score élevé pour l'historique d'un jeu, par exemple, vous devrez stocker ces données afin qu'elles soient disponibles la prochaine fois que quelqu'un jouera au jeu. Les données conservées même après la fermeture d'une application sont appelées données persistantes et sont stockées dans un type de base de données.

Nous explorerons l'utilisation de la mémoire à court terme (variables) et à long terme mémoire (données de la base de données) aux chapitres 16 et 22, respectivement.

Les gestionnaires d'événements peuvent interagir avec le Web

Certaines applications utilisent uniquement les informations contenues dans le téléphone ou l'appareil. Mais de nombreuses applications communiquent avec le Web, soit en affichant une page Web au sein de l'application, soit en envoyant des requêtes aux API de services Web (interfaces de programmation d'applications). De telles applications sont dites « compatibles avec le Web ».

Twitter est un exemple de service Web avec lequel une application App Inventor peut communiquer. Vous pouvez écrire des applications qui demandent et affichent les tweets précédents de vos amis et également mettre à jour votre statut Twitter. Les applications qui communiquent avec plusieurs services Web sont appelées mashups. Nous explorerons les applications Web au chapitre 24.

Résumé

Un créateur d'application doit voir son application à la fois du point de vue de l'utilisateur final et du point de vue approfondi d'un programmeur. Avec App Inventor, vous concevez l'apparence d'une application et son comportement, c'est-à-dire l'ensemble des gestionnaires d'événements qui permettent à une application de se comporter comme vous le souhaitez. Vous créez ces gestionnaires d'événements en assemblant et en configurant des blocs représentant des événements, des fonctions, des branches conditionnelles, des boucles de répétition, des appels Web, des opérations de base de données, etc., puis testez votre travail en exécutant réellement l'application sur votre téléphone. Après avoir écrit quelques programmes, la correspondance entre la structure interne d'une application et sa manifestation physique devient claire. Lorsque cela arrive, vous êtes un programmeur !

Ingénierie et débogage d'une application

HelloPurr, MoleMash et les autres applications

abordées dans les premiers chapitres de ce livre sont des projets logiciels relativement petits et ne nécessitent pas vraiment une quantité importante d'ingénierie logicielle.

Dès que vous entreprenez un projet plus complexe, vous vous rendrez compte que la difficulté de créer un logiciel augmente rapidement pour chaque élément de complexité que vous ajoutez – ce n'est pas du tout proche d'une relation linéaire.

Vous apprendrez rapidement que pour créer un logiciel, même modérément complexe, vous avez besoin de prévoyance, de planification, de plans, de tests utilisateur et système et, en général, de techniques et de compétences qui relèvent davantage de l'ingénierie que de la programmation. Pour la plupart d'entre nous, il faut quelques coups durs avant de réaliser ce fait. À ce stade, vous serez prêt à apprendre quelques principes de génie logiciel et techniques de débogage. Si vous en êtes déjà à ce stade, ou si vous faites partie de ces rares personnes qui souhaitent apprendre quelques techniques dans l'espoir d'éviter certaines de ces douleurs de croissance, ce chapitre est pour vous.

Figure 15-1.



Principes du génie logiciel

Voici quelques principes de base que nous aborderons dans ce chapitre :

- Impliquez vos utilisateurs potentiels dans le processus le plus tôt et le plus souvent possible.
- Construisez un premier prototype simple, puis complétez-le progressivement.
- Codez et testez par petits incrément, jamais plus de quelques blocs à la fois.
- Concevez la logique de votre application avant de commencer à coder.
- Divisez, superposez et conquérez.
- Commentez vos blocages afin que les autres (et vous) puissiez les comprendre.
- Apprenez à tracer des blocs avec un crayon et du papier afin de comprendre leur mécanique.

Si vous suivez ces conseils, vous vous épargnerez du temps et de la frustration et construirez mieux logiciel. Mais vous ne le suivrez probablement pas à chaque fois ! Certains de ces conseils pourraient

semblent contre-intuitifs. Votre tendance naturelle est de penser à une idée, de supposer que vous savez ce que veulent vos utilisateurs, puis de commencer à assembler des blocs jusqu'à ce que vous pensiez avoir terminé l'application. Revenons au premier principe et voyons comment comprendre ce que veulent vos utilisateurs avant de commencer à créer quoi que ce soit.

Résoudre de vrais problèmes

Dans le film *Field of Dreams*, le personnage de Ray entend une voix murmurer : « Si vous le construisez, [ils] viendront. » Ray écoute les murmures, construit un terrain de baseball au milieu de son champ de maïs de l'Iowa et, en effet, les White Sox de 1919 et des milliers de fans se présentent.

Il faut savoir dès maintenant que les conseils du chuchoteur ne s'appliquent pas aux logiciels.

En fait, c'est le contraire de ce que vous devriez faire. L'histoire du logiciel est parsemée d'excellentes solutions pour lesquelles il n'y a aucun problème. Résoudre un vrai problème est ce qui fait une application étonnante et un projet réussi et peut-être lucratif. Et pour savoir quel est le problème, il faut en parler aux personnes qui en sont atteintes. Ceci est souvent appelé conception centrée sur l'utilisateur et cela vous aidera à créer de meilleures applications.

Si vous rencontrez des programmeurs, demandez-leur quel pourcentage des programmes qu'ils ont écrits ont réellement été déployés auprès de vrais utilisateurs. Vous serez surpris de voir à quel point ce pourcentage est faible, même pour les grands programmeurs. La plupart des projets logiciels rencontrent tellement de problèmes qu'ils ne voient jamais le jour.

La conception centrée sur l'utilisateur signifie réfléchir et discuter tôt et souvent avec les utilisateurs potentiels. En réalité, cela devrait commencer avant même que vous décidiez quoi construire. La plupart des logiciels les plus performants ont été conçus pour résoudre le problème d'une personne en particulier, puis – et alors seulement – généralisés pour devenir la prochaine grande nouveauté.

Construire un prototype et montrer aux utilisateurs

La plupart des utilisateurs potentiels ne fourniront pas de commentaires utiles si vous leur demandez de lire un document spécifiant ce que l'application fera et de donner leur avis sur cette base.

Ce qui fonctionne, c'est de leur montrer un modèle interactif pour l'application que vous allez créer : un prototype. Un prototype est une version incomplète et non référencée de l'application.

Lorsque vous le construisez, ne vous souciez pas des détails, de l'exhaustivité ou d'une belle interface graphique ; construisez-le de manière à ce qu'il fasse juste assez pour illustrer la valeur fondamentale de l'application. Ensuite, montrez-le à vos utilisateurs potentiels, restez silencieux et écoutez.

Développement incrémental

Lorsque vous démarrez votre première application de taille importante, votre inclination naturelle pourrait être d'ajouter tous les composants et blocs dont vous aurez besoin en un seul grand effort, puis de télécharger l'application sur votre téléphone pour voir si elle fonctionne. Prenez, par exemple, une application de quiz.

Sans conseils, la plupart des programmeurs débutants ajouteront des blocs avec une longue liste de questions et de réponses, des blocs pour gérer la navigation dans le quiz, des blocs pour gérer la vérification de la réponse de l'utilisateur et des blocs pour chaque détail de la logique de l'application, le tout avant de tester pour voir si tout cela fonctionne. En génie logiciel, c'est ce qu'on appelle l'approche Big Bang.

Presque tous les nouveaux programmeurs utilisent cette approche. Dans mes cours (de l'auteur Wolber) à l'Université de San Francisco, je demande souvent à un étudiant : « Comment ça va ? lorsque l'étudiant travaille sur une application.

«Je pense que j'ai fini», répondra l'étudiant.

"Splendide. Puis je le voir?"

« Euh, pas encore ; Je n'ai pas mon téléphone avec moi.

« Donc, vous n'avez pas du tout exécuté l'application ? » Je demande.

"Non."

Je regarderai par-dessus l'épaule de l'élève une configuration étonnante et colorée de 30 ou donc des blocs, aucun d'entre eux n'a été testé. Le problème c'est que quand on teste tout d'un coup, il est beaucoup plus difficile de diagnostiquer les bugs, et il y aura des bugs, des gros poilus !

Probablement le meilleur conseil que je puisse donner à mes étudiants et aux futurs programmeurs.

partout, est-ce que c'est :

Codez un peu, testez un peu, répétez.

Créez votre application une pièce à la fois, en la testant au fur et à mesure. Vous trouverez des bugs, d'accord, mais les plus petits que vous pouvez facilement écraser. Et le processus deviendra étonnamment satisfaisant, car vous verrez des résultats plus tôt lorsque vous le suivrez.

Des centaines de livres et de thèses ont été rédigés sur des logiciels incrémentaux développement. Si vous êtes intéressé par le processus de création de logiciels (et d'autres choses), consultez la méthodologie de développement agile.¹

Concevoir avant de coder

La programmation comporte deux parties : comprendre la logique de l'application, puis traduire cette logique en code dans un langage de programmation. Avant de vous lancer dans la traduction, consacrez un peu de temps à la logique. Spécifiez ce qui doit se passer à la fois pour l'utilisateur et en interne dans l'application. Déterminez la logique de chaque gestionnaire d'événements avant de passer à la traduction de cette logique en blocs.

Des livres entiers ont été écrits sur diverses méthodologies de conception de programmes. Certaines personnes utilisent des diagrammes tels que des diagrammes ou des organigrammes pour la conception, tandis que d'autres préfèrent le texte et les croquis manuscrits. Certaines personnes pensent que tout « design » devrait

¹ Beck, Kent ; et coll. (2001). « Manifeste pour le développement logiciel agile ». Agile Alliance, récupéré le 5 juin 2014

se retrouver directement à côté de votre code sous forme d'annotation (commentaires), pas dans un document séparé. La clé pour les débutants est de comprendre qu'il existe une logique dans tous les programmes qui n'a rien à voir avec un langage de programmation particulier. Aborder simultanément cette logique et sa traduction dans une langue, aussi intuitive soit-elle, peut s'avérer écrasant. Ainsi, tout au long du processus, éloignez-vous de l'ordinateur et réfléchissez à votre application, assurez-vous de bien comprendre ce que vous voulez qu'elle fasse et documentez ce que vous proposez d'une manière ou d'une autre. Ensuite, assurez-vous de connecter cette documentation de conception à votre application afin que d'autres puissent en bénéficier.

Commentez votre code

Si vous avez suivi quelques didacticiels de ce livre, vous avez probablement vu les petites cases jaunes à l'intérieur des blocs (voir Figure 15-1). C'est ce qu'on appelle des commentaires. Dans App Inventor, vous pouvez ajouter des commentaires à n'importe quel bloc en cliquant dessus avec le bouton droit et en choisissant Ajouter un commentaire. Les commentaires ne sont que des annotations ; ils n'affectent pas du tout l'exécution de l'application.



Figure 15-2. Utiliser un commentaire sur le bloc if pour décrire ce qu'il fait en anglais simple

Pourquoi commenter, alors ? Eh bien, si votre application réussit, elle vivra longtemps. Même après avoir passé seulement une semaine loin de votre application, vous oublierez ce que vous pensiez à ce moment-là et n'aurez aucune idée de la raison pour laquelle certains blocs sont utilisés. Pour cette raison, même si personne d'autre ne verra jamais vos blocs, vous devez les commenter.

Et si votre application réussit, elle passera sans doute entre plusieurs mains. Personnes voudra le comprendre, le personnaliser et l'étendre. Dès que vous vivrez la merveilleuse expérience de démarrer un projet avec le code non commenté de quelqu'un, vous comprendrez parfaitement pourquoi les commentaires sont essentiels.

Annoter un programme n'est pas intuitif et je n'ai jamais rencontré de programmeur débutant qui pensait que c'était important. À l'inverse, je n'ai jamais rencontré de programmeur expérimenté qui ne l'ait pas fait.

Diviser, superposer et conquérir

Les problèmes deviennent insurmontables lorsqu'ils sont trop importants. La clé est de résoudre un problème. Il existe deux manières principales de procéder. Celui que nous connaissons le mieux est de briser

un problème en parties (A, B, C) et abordez chacune individuellement. Une deuxième méthode, moins courante, consiste à diviser un problème en couches allant du simple au complexe. Ajoutez quelques blocs pour un comportement simple, testez le logiciel pour vérifier qu'il se comporte comme vous le souhaitez, puis ajoutez une autre couche de complexité, et ainsi de suite.

En utilisant l'application President's Quiz du chapitre 10 comme exemple, évaluons ces deux méthodes. Rappelons que l'application President's Quiz permet à l'utilisateur de parcourir les questions en cliquant sur un bouton Suivant. Il vérifie également les réponses de l'utilisateur pour déterminer si elle a raison. Ainsi, lors de la conception de cette application, vous pouvez la diviser en deux parties : la navigation dans les questions et la vérification des réponses, et programmer chacune séparément.

Cependant, au sein de chacune de ces deux parties, vous pouvez également décomposer le processus du simple au complexe. Ainsi, pour la navigation dans les questions, commencez par créer le code pour afficher uniquement la première question de la liste des questions, et testez-le pour vous assurer qu'il fonctionne. Ensuite, créez le code pour passer à la question suivante, mais ignorez le problème de ce qui se passe lorsque vous arrivez à la dernière question. Après avoir confirmé que le quiz vous mènera jusqu'au bout, ajoutez les blocs pour gérer le « cas particulier » de l'utilisateur atteignant la dernière question.

Il ne s'agit pas de savoir si vous devez diviser un problème en plusieurs parties, ou en couches de complexité : vous devriez faire les deux. Ceux qui savent bien faire cela, les architectes logiciels, sont extrêmement demandés.

Comprenez votre langue : suivi avec un stylet et Papier

Lorsqu'une application est en action, elle n'est que partiellement visible. L'utilisateur final d'une application ne voit que sa face extérieure, les images et les données affichées dans l'interface utilisateur. Le fonctionnement interne des logiciels est caché au monde extérieur, tout comme les mécanismes internes du cerveau humain (heureusement !). Lors de l'exécution d'une application, nous ne voyons pas les instructions (blocs), nous ne voyons pas le compteur de programme qui suit quelle instruction est en cours d'exécution et nous ne voyons pas les cellules de mémoire interne du logiciel (ses variables et propriétés)... En fin de compte, c'est ainsi que nous le souhaitons : l'utilisateur final ne doit voir que ce que le programme affiche explicitement. Cependant, pendant que vous développez et testez un logiciel, vous souhaitez voir tout ce qui se passe.

Vous, le programmeur, voyez le code pendant le développement, mais seulement une vue statique de il. Ainsi, vous devez imaginer le logiciel en action : des événements se produisant, le compteur du programme se déplaçant et exécutant le bloc suivant, les valeurs dans les cellules mémoire changeant, etc.

La programmation nécessite un changement entre deux points de vue différents. Vous commencez avec le modèle statique (les blocs de code) et essayez d'imaginer comment le programme se comportera réellement. Lorsque vous êtes prêt, vous passez en mode test : vous jouez le rôle de l'utilisateur final et testez le logiciel pour voir s'il se comporte comme prévu. Si ce n'est pas le cas, vous

devez revenir à la vue statique, peaufiner votre modèle et tester à nouveau. Grâce à ce processus de va-et-vient, vous avancez vers une solution acceptable.

Lorsque vous commencez à programmer, vous ne disposez que d'un modèle partiel de la façon dont un ordinateur le programme fonctionne – l'ensemble du processus semble presque magique. Vous commencez avec quelques applications simples : cliquer sur un bouton fait miauler un chat ! Vous passez ensuite à des applications plus complexes, parcourez quelques didacticiels et apportez peut-être quelques modifications pour les personnaliser. Le débutant comprend en partie le fonctionnement interne des applications, mais ne se sent certainement pas maître du processus. Le débutant dira souvent : « Ça ne marche pas » ou « Ça ne fait pas ce qu'il est censé faire ». La clé est d'apprendre comment les choses fonctionnent au point de penser le programme de manière plus subjective et de dire des choses telles que « Mon programme fait ceci » et « Ma logique fait que le programme... »

Une façon d'apprendre comment fonctionnent les programmes consiste à retracer l'exécution d'une application simple, représenter sur papier exactement ce qui se passe à l'intérieur de l'appareil lorsque chaque bloc est exécuté. Imaginez l'utilisateur déclenchant un gestionnaire d'événements, puis parcourez et montrez l'effet de chaque bloc : comment les variables et les propriétés de l'application changent-elles ? Comment les composants de l'interface utilisateur changent-ils ? Comme une lecture attentive dans un cours de littérature, ce traçage étape par étape vous oblige à examiner les éléments du langage – dans ce cas, les blocs App Inventor.

La complexité de l'échantillon que vous tracez est presque sans importance ; la clé est de ralentir votre processus de réflexion et d'examiner la cause et l'effet de chaque blocage. Vous commencerez progressivement à comprendre que les règles régissant l'ensemble du processus ne sont pas aussi écrasantes que vous le pensiez au départ.

Par exemple, considérons les blocs représentés à la figure 15-2, qui sont légèrement modifications de celles de l'application President's Quiz (Chapitre 8).

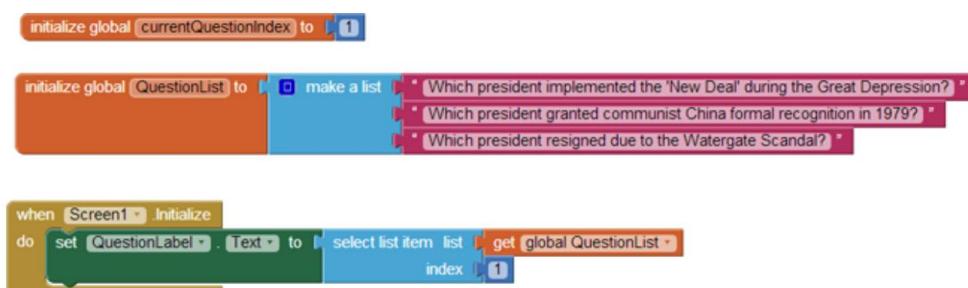


Figure 15-3. Définition du texte de QuestionLabel sur le premier élément de QuestionList au démarrage de l'application

Comprenez-vous ce code ? Pouvez-vous le retracer et montrer exactement ce qui se passe à chaque étape ?

Vous commencez à tracer en dessinant d'abord des cases de cellules mémoire pour toutes les variables pertinentes et propriétés. Dans ce cas, vous avez besoin de cases pour currentQuestionIndex et QuestionLabel.Text, comme indiqué dans le tableau 15-1.

Tableau 15-1. Boîtes à cellules mémoire pour le traçage

QuestionLabel.Text	currentQuestionIndex

Ensuite, réfléchissez à ce qui se passe lorsqu'une application démarre, non pas du point de vue de l'utilisateur, mais en interne, au sein de l'application lors de son initialisation. Si vous avez suivi certains tutoriels, vous le savez probablement, mais peut-être n'y avez-vous pas pensé en termes mécaniques. Lorsqu'une application démarre :

1. Toutes les propriétés des composants sont définies en fonction de leurs valeurs initiales dans le Concepteur de composants.
2. Toutes les définitions et initialisations de variables sont effectuées.
3. Les blocs du gestionnaire d'événements Screen.Initialize sont exécutés.

Tracer un programme vous aide à comprendre ces mécanismes. Alors, que devrait-il y avoir les cases après la phase d'initialisation ?

Comme le montre le tableau 15-2, le 1 est dans currentQuestionIndex car la définition de variable est exécutée au démarrage de l'application et elle l'initialise à 1. La première question est dans QuestionLabel.Text car Screen.Initialize sélectionne le premier élément de QuestionList et le met là.

Tableau 15-2. Les valeurs après l'initialisation de l'application President's Quiz

QuestionLabel.Text	indexquestionactuelle
Quel président a mis en œuvre le « New Deal » pendant la Grande Dépression ? 1	

Ensuite, tracez ce qui se passe lorsque l'utilisateur clique sur le bouton Suivant, en utilisant les blocs illustré à la figure 15-3.

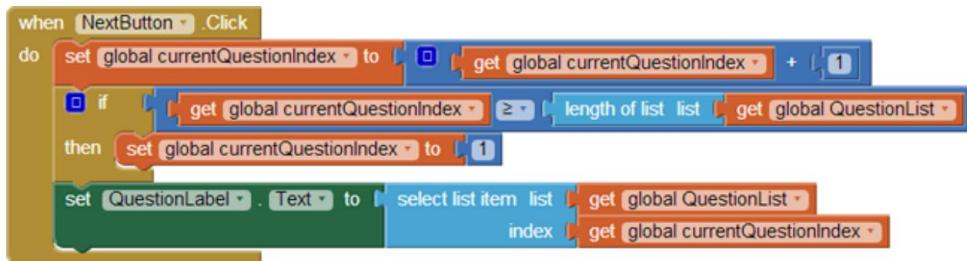


Figure 15-4. Ce bloc est exécuté lorsque l'utilisateur clique sur NextButton

Examinez chaque bloc, un par un. Tout d'abord, le currentQuestionIndex est incrémenté.

À un niveau encore plus détaillé, la valeur actuelle de la variable (1) est ajoutée à 1 et le résultat (2) est placé dans currentQuestionIndex. L'instruction if est fausse car la valeur de currentQuestionIndex (2) est inférieure à la longueur de QuestionList (3).

Par conséquent, le deuxième élément est sélectionné et placé dans QuestionLabel.Text, comme illustré dans le tableau 15-3.

Tableau 15-3. Les valeurs après avoir cliqué sur NextButton

QuestionLabel.Text	indexquestionactuelle
Quel président a reconnu officiellement la Chine communiste en 1979 ? 2	

Suivez ce qui se passe au deuxième clic. Maintenant, currentQuestionIndex est incrémenté et devient 3. Que se passe-t-il avec le if ? Avant de lire, examinez-le de très près et voyez si vous pouvez le retracer correctement.

Sur le test if, la valeur de currentQuestionIndex (3) est en effet supérieure ou égale à la longueur de QuestionList. Par conséquent, currentQuestionIndex est défini sur 1 et la première question est placée dans l'étiquette, comme indiqué dans le tableau 15-4.

Tableau 15-4. Les valeurs après que NextButton soit cliqué une deuxième fois

QuestionLabel.Text	indexquestionactuelle
Quel président a mis en œuvre le « New Deal » pendant la Grande Dépression ? 1	

La trace a mis en évidence un bug : la dernière question de la liste n'apparaît jamais ! Est-ce que tu sais comment le réparer ?

Lorsque vous pouvez retracer une application à ce niveau de détail, vous devenez un programmeur, un ingénieur. Vous commencez à comprendre les mécanismes du langage de programmation, en absorbant des phrases et des mots dans le code au lieu de saisir vaguement les paragraphes.

Oui, le langage de programmation est complexe, mais chaque « mot » a une interprétation définie et simple par la machine. Si vous comprenez comment chaque bloc correspond à une variable ou à une modification de propriété, vous pouvez comprendre comment écrire ou modifier votre application. Vous réalisez que vous avez le contrôle total.

Maintenant, si vous disiez à vos amis : « J'apprends à permettre à un utilisateur de cliquer sur un bouton Suivant pour passer à la question suivante ; c'est vraiment dur », ils penseraient que tu es fou. En fait, une telle programmation est très difficile, non pas parce que les concepts sont si complexes, mais parce que vous devez ralentir votre cerveau pour comprendre comment lui-même, ou un ordinateur, traite chaque étape, y compris les choses que votre cerveau fait inconsciemment.

Débogage d'une application

Tracer une application étape par étape, sur papier, est une façon de comprendre la programmation ; c'est également une méthode éprouvée pour déboguer une application lorsqu'elle rencontre des problèmes.

Les environnements de programmation, notamment App Inventor, fournissent également la version haute technologie du traçage au stylo et au papier grâce à des outils de débogage qui automatisent une partie du processus. De tels outils améliorent le processus de développement d'applications en fournissant une vue éclairée d'une application en action. Ces outils permettent au programmeur d'effectuer les opérations suivantes :

- Suspendez une application à tout moment et examinez ses variables et propriétés.
- Exécuter des instructions individuelles (blocs) pour examiner leurs effets

Surveillance des variables

Les valeurs des propriétés et des variables des composants ne sont pas visibles lorsque vous testez une application dans App Inventor. Une technique de débogage courante consiste à ajouter des blocs pour afficher ces valeurs dans les étiquettes de l'interface utilisateur pendant les tests, puis à supprimer les étiquettes et à afficher le code une fois l'application déboguée.

La version antérieure d'App Inventor (App Inventor Classic) disposait d'un mécanisme permettant de surveiller les valeurs des variables et des propriétés dans l'éditeur de blocs pendant les tests, sans utiliser d'étiquettes dans l'interface utilisateur. Il est prévu qu'un tel mécanisme soit également ajouté à App Inventor 2, alors restez à l'affût car il est très utile pour le débogage et la compréhension du code.

Test des blocs individuels

Bien que vous puissiez utiliser le mécanisme Watch pour examiner des variables lors de l'exécution d'une application, un autre outil appelé Do It vous permet d'essayer des variables individuelles.

blocs en dehors de la séquence d'exécution ordinaire. Cliquez avec le bouton droit sur n'importe quel bloc et choisissez Do It ; le bloc sera exécuté. Si le bloc est une expression qui renvoie une valeur, App Inventor affichera cette valeur dans une zone au-dessus du bloc.

Do It est très utile pour déboguer les problèmes de logique dans vos blocs. Considérez à nouveau le gestionnaire d'événements NextButton.Click du quiz et supposons qu'il présente un problème de logique dans lequel vous ne parcourez pas toutes les questions. Vous pouvez tester le programme en cliquant sur Suivant dans l'interface utilisateur et en vérifiant si la question appropriée apparaît à chaque fois. Vous pouvez même regarder le currentQuestionIndex pour voir comment chaque clic le modifie.

Malheureusement, ce type de test vous permet uniquement d'examiner l'effet de gestionnaires d'événements entiers. L'application effectuera tous les blocs dans le gestionnaire d'événements pour le clic sur le bouton avant de vous permettre d'examiner vos variables Watch ou l'interface utilisateur.

Avec l'outil Do It, vous pouvez ralentir le processus de test et examiner l'état de l'application après tout blocage. Le schéma général consiste à lancer des événements d'interface utilisateur jusqu'à ce que vous arriviez au point problématique dans l'application. Après avoir découvert que la troisième question n'apparaissait pas dans l'application de quiz, vous pouvez cliquer une fois sur le bouton Suivant pour accéder à la deuxième question. Ensuite, au lieu de cliquer à nouveau sur NextButton et d'exécuter l'intégralité du gestionnaire d'événements d'un seul coup, vous pouvez utiliser Do It pour exécuter les blocs dans le gestionnaire d'événements NextButton.Click , un à la fois. Vous commenceriez par cliquer avec le bouton droit sur la rangée supérieure de blocs (l'incrément de currentQuestionIndex) et en choisissant Do It, comme illustré dans la figure 15-4.

Cela changerait l'index à 3. L'exécution de l'application s'arrêterait alors : Do It entraîne l'exécution uniquement du bloc choisi et de tous les blocs subordonnés. Cela vous offre, à vous le testeur, la possibilité d'examiner les variables surveillées et l'interface utilisateur. Lorsque vous êtes prêt, vous pouvez choisir la rangée de blocs suivante (le test if) et sélectionner Do It pour qu'il soit exécuté. À chaque étape du processus, vous pouvez voir l'effet de chaque bloc.

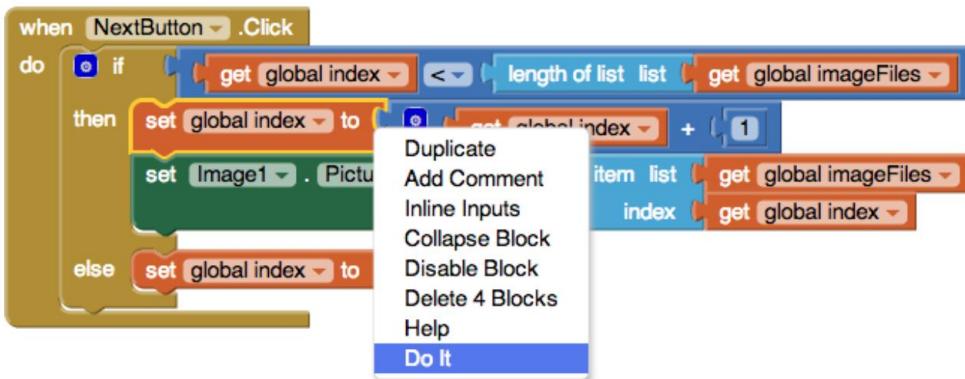


Figure 15-5. Utiliser l'outil Do It pour exécuter les blocs un par un

Développement incrémental avec Do It

Il est important de noter que l'exécution de blocs individuels ne sert pas uniquement au débogage. Vous pouvez également l'utiliser pendant le développement pour tester les blocs au fur et à mesure. Par exemple, si vous créez une longue formule pour calculer la distance en miles entre deux coordonnées GPS, vous pourriez tester la formule à chaque étape pour vérifier que les blocs font sens.

Désactiver les blocages

Une autre façon de vous aider à déboguer et tester votre application de manière incrémentielle consiste à désactiver les blocages. En faisant cela, vous pouvez laisser des blocs problématiques ou non testés dans une application, mais demander au système de les ignorer temporairement pendant l'exécution de l'application. Vous pouvez ensuite tester les blocs actifs et les faire fonctionner pleinement sans vous soucier des blocs problématiques. Vous pouvez désactiver n'importe quel blocage en cliquant dessus avec le bouton droit et en choisissant Désactiver le blocage. Le bloc sera grisé et lorsque vous exécuterez l'application, il sera ignoré. Lorsque vous êtes prêt, vous pouvez activer le blocage en cliquant à nouveau dessus avec le bouton droit et en choisissant Activer le blocage.

Résumé

L'avantage d'App Inventor est sa simplicité. Sa nature visuelle vous permet de commencer immédiatement à créer une application et vous n'avez pas à vous soucier de nombreux détails de bas niveau. Mais la réalité est qu'App Inventor ne peut pas déterminer ce que votre application devrait faire pour vous, et encore moins exactement comment le faire. Même s'il est tentant de se lancer directement dans le concepteur et l'éditeur de blocs et de commencer à créer une application, il est important de consacrer du temps à réfléchir et à planifier en détail ce que fera exactement votre application. Cela semble un peu pénible, mais si vous écoutez vos utilisateurs, prototyppez, testez et tracez la logique de votre application, vous créerez de meilleures applications en un rien de temps.

Programmation de la mémoire de votre application

Tout comme les gens ont besoin de se souvenir des choses, les applications aussi. Ce chapitre examine comment programmer une application pour mémoriser des informations.

Lorsque quelqu'un vous donne le numéro de téléphone d'une pizzeria, votre cerveau le stocke dans un emplacement mémoire. Si quelqu'un appelle des nombres pour que vous les ajoutiez, vous stockez les nombres et les résultats intermédiaires dans des emplacements mémoire. Dans de tels cas, vous n'êtes pas pleinement conscient de la manière dont votre cerveau stocke les informations ou les mémorise.

Une application a aussi une mémoire, mais son fonctionnement interne est bien moins mystérieux que celui de votre cerveau. Dans ce chapitre, vous apprendrez comment configurer la mémoire d'une application, comment y stocker des informations et comment récupérer ces informations ultérieurement.

Emplacements de mémoire nommés

La mémoire d'une application se compose d'un ensemble d'emplacements mémoire nommés. Certains de ces emplacements mémoire sont créés lorsque vous faites glisser un composant dans votre application ; ces emplacements sont appelés propriétés. Vous pouvez également définir des emplacements mémoire nommés qui ne sont pas associés à un composant particulier ; c'est ce qu'on appelle des variables. Alors que les propriétés sont généralement associées à ce qui est visible dans une application, les variables peuvent être considérées comme la mémoire « scratch » cachée de l'application.

Propriétés

L'utilisateur d'une application peut voir les composants visibles tels que les boutons, les zones de texte et les étiquettes. Cependant, en interne, chaque composant est entièrement défini par un ensemble de propriétés. Les valeurs stockées dans les emplacements mémoire de chaque propriété déterminent la façon dont le composant apparaît.

Vous définissez les valeurs des propriétés directement dans le Concepteur de composants. Par exemple, La figure 16-1 montre le panneau permettant de modifier les propriétés d'un composant Canvas .

Figure 16-1.



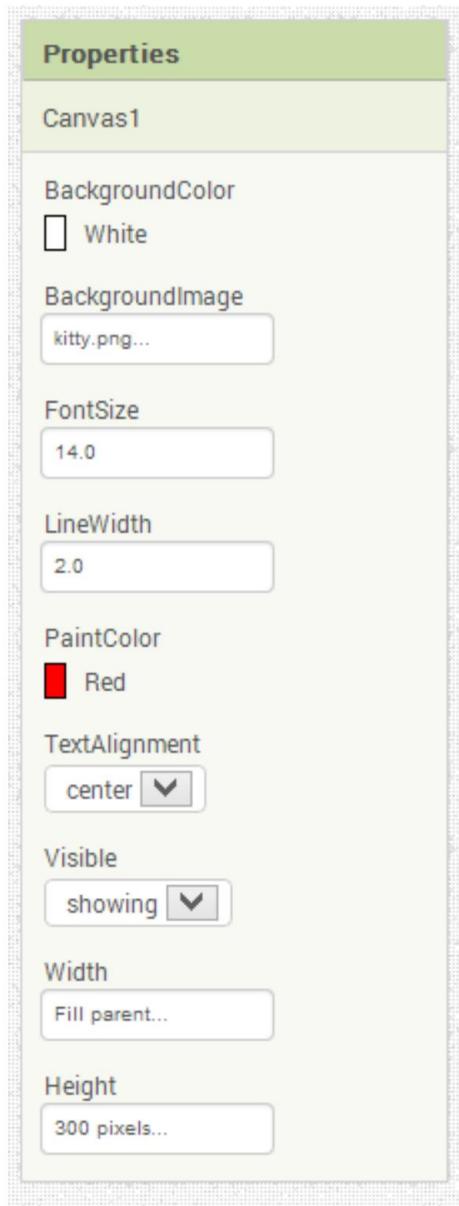


Figure 16-2. Vous pouvez définir les propriétés des composants dans Designer ; vous définissez les valeurs initiales des propriétés (elles n'affichent pas les valeurs actuelles lors de l'exécution d'une application)

Le composant Canvas possède de nombreuses propriétés de différents types. Par exemple, le BackgroundColor et PaintColor sont des emplacements mémoire contenant une couleur.

Le BackgroundImage contient un nom de fichier (kitty.png). La propriété Visible contient un booléen

valeur (vrai ou faux selon que la case est cochée ou non). Les emplacements Largeur et Hauteur contiennent un numéro ou une désignation spéciale (par exemple, « Fill parent »).

Lorsque vous définissez une propriété dans le Concepteur de composants, vous spécifiez la valeur initiale valeur de la propriété : sa valeur au premier démarrage de l'application. Les valeurs des propriétés peuvent également être modifiées au fur et à mesure de l'exécution de l'application, avec des blocs. Pourtant, les valeurs affichées dans Component Designer, telles que celles de la figure 16-1, ne changent pas ; ceux-ci affichent toujours uniquement les valeurs initiales. Cela peut prêter à confusion lorsque vous testez une application : la valeur actuelle des propriétés de l'application n'est pas visible.

Définition des variables

Comme les propriétés, les variables sont nommées emplacements mémoire, mais elles ne sont pas associées à un composant particulier. Vous définissez une variable lorsque votre application doit mémoriser quelque chose qui n'est pas stocké dans une propriété de composant. Par exemple, une application de jeu peut avoir besoin de mémoriser le niveau atteint par l'utilisateur. Si le numéro de niveau devait apparaître dans un composant Label , vous n'aurez peut-être pas besoin d'une variable, car vous pourriez simplement stocker le niveau dans la propriété Text du composant Label .

Mais si le numéro de niveau n'est pas quelque chose que l'utilisateur verra, vous définirez une variable dans laquelle le stocker.

Le Quiz des Présidents (Chapitre 8) est un autre exemple d'application nécessitant une variable.

Dans cette application, une seule question du quiz doit apparaître à la fois dans l'interface utilisateur, mais le quiz comporte de nombreuses questions (dont la plupart sont cachées à l'utilisateur à tout moment). Il faut donc définir une variable pour stocker la liste des questions.

Alors que les propriétés sont créées automatiquement lorsque vous faites glisser un composant dans le Designer, les variables sont définies explicitement dans l'éditeur de blocs en faisant glisser un bloc global d'initialisation . Vous pouvez nommer la variable en cliquant sur le texte « nom » dans le bloc, et vous pouvez lui spécifier une valeur initiale en faisant glisser un nombre, un texte, une couleur ou en créant un bloc de liste et en le branchant. Voici les étapes à suivre. suivrait pour créer une variable appelée score avec une valeur initiale de 0 :

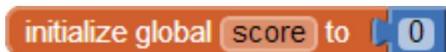
1. Dans Blocs intégrés, ouvrez le tiroir Variables et faites glisser le bloc global d'initialisation .



2. Modifiez le nom de la variable en cliquant sur le texte « nom » et en tapant "score".



3. Depuis le tiroir Math, faites glisser un bloc numérique et branchez-le dans le socket de la définition de la variable pour définir la valeur initiale.



Lorsque vous définissez une variable, vous demandez à l'application de configurer un emplacement mémoire nommé pour stocker une valeur. Ces emplacements mémoire, comme les propriétés, ne sont pas visibles par l'utilisateur lorsque l'application s'exécute.

Le bloc numérique que vous branchez spécifie la valeur qui doit être placée dans l'emplacement au démarrage de l'application. En plus de linitialisation avec des nombres ou du texte, vous pouvez également initialiser la variable en créant une liste ou en créant un bloc de liste vide . Cela informe lapplication que la variable stockera une liste demplacements mémoire au lieu d'une valeur unique. Pour en savoir plus sur les listes, consultez le chapitre 19.

Définir et obtenir une variable

Lorsque vous définissez une variable, App Inventor crée deux blocs pour celle-ci : set et get. Vous pouvez accéder à ces blocs en survolant le nom de la variable dans le bloc dinitialisation, comme illustré dans la figure 16-2.



Figure 16-3. Le bloc dinitialisation contient des blocs set et get pour cette variable

Le set global to block vous permet de modifier la valeur stockée dans la variable. Par exemple, le bloc numérique de la figure 16-3 place la valeur 5 dans la variable score. Le terme « global » dans le score global défini à bloquer fait référence au fait que la variable peut être utilisée dans tous les gestionnaires d'événements et procédures du programme. Avec la dernière version dApp Inventor, vous pouvez également définir des variables « locales » pour une procédure ou un gestionnaire d'événements particulier. Autrement dit, les variables locales ne peuvent être utilisées que par la procédure ou l'événement auquel elles sont associées (plus d'informations à ce sujet). un peu plus loin dans le chapitre).

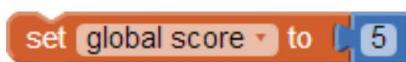


Figure 16-4. Placer un chiffre 5 dans le score variable

Vous utilisez le bloc intitulé obtenir le score global pour récupérer la valeur d'une variable. Par exemple, si vous souhaitez vérifier si la valeur à l'intérieur de l'emplacement mémoire est supérieure à

100, vous brancheriez le bloc get global score dans un test if , comme le montre la figure 16-4.



Figure 16-5. Utiliser le bloc de score global pour obtenir la valeur stockée dans la variable

Définition d'une variable sur une expression

Comme vous l'avez vu, vous pouvez mettre des valeurs simples telles que 5 dans une variable, mais vous définirez souvent la variable sur une expression plus complexe (l'expression est le terme informatique désignant une formule). Par exemple, lorsque l'utilisateur clique sur Suivant pour passer à la question suivante dans une application de quiz, vous devez définir la variable currentQuestion sur une valeur supérieure à sa valeur actuelle. Lorsqu'une personne perd dix points dans une application de jeu, vous devez modifier la variable de score à 10 de moins que sa valeur actuelle. Dans un jeu comme MoleMash (Chapitre 3), vous modifiez l'emplacement horizontal (x) de la taupe en une position aléatoire dans une toile.

Vous construirez de telles expressions avec un ensemble de blocs qui se connectent à un ensemble global à bloquer.

Incrémenter une variable

L'expression la plus courante consiste peut-être à incrémenter une variable ou à définir une variable en fonction de sa propre valeur actuelle. Par exemple, dans un jeu, lorsqu'un joueur marque un point, le score variable peut être incrémenté de 5. La figure 16-5 montre les blocs permettant d'implémenter ce comportement.



Figure 16-6. Incrémenter le score variable de 5

Si vous parvenez à comprendre ce genre de blocages, vous êtes sur la bonne voie pour devenir un programmeur. Vous lisez ces blocs comme « définissez le score à cinq fois plus qu'il ne l'est déjà », ce qui est une autre façon de dire incrémenter votre variable. La façon dont cela fonctionne est que les blocs sont interprétés à l'envers, et non de gauche à droite. Ainsi, les blocs les plus internes – le score global d'obtention et le bloc numéro 5 – sont évalués en premier. Ensuite, le bloc + est exécuté et le résultat est « défini » dans le score variable.

Supposons qu'il y ait un 10 dans l'emplacement mémoire pour le score avant ces blocs ; le L'application effectuerait les étapes suivantes :

1. Récupérez le 10 de l'emplacement mémoire du score (évaluez le bloc get).
2. Ajoutez-y 5 pour obtenir 15.
3. Placez le résultat, 15, dans l'emplacement mémoire de la partition (exécutant l' ensemble).

Création d'expressions complexes

Dans le tiroir Mathématiques, App Inventor propose un large éventail de fonctions mathématiques similaires à celles que vous trouverez dans une feuille de calcul ou une calculatrice. Il existe des opérateurs arithmétiques (par exemple +, -, *, /), des blocs pour générer des valeurs aléatoires et des opérateurs tels que sqrt, cosinus et sinus.

Vous pouvez utiliser ces blocs pour créer une expression complexe, puis les connecter comme l'expression de droite d'un ensemble global à bloquer. Par exemple, pour déplacer un sprite d'image vers une colonne aléatoire dans les limites d'un canevas, vous devez configurer une expression composée d'un bloc de multiplication (*), d'un bloc de soustraction (-), d'une propriété Canvas1.Width et d'un ImageSprite1. Propriété de largeur et un bloc de fractions aléatoires , comme illustré dans la figure 16-6.

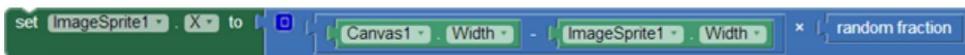


Figure 16-7. Vous pouvez utiliser des blocs mathématiques pour créer des expressions complexes comme celle-ci

Comme pour l'exemple d'incrémentation de la section précédente, les blocs sont interprétés par l'application de manière inversée. Supposons que le canevas ait une largeur de 300 et que l' ImageSprite ait une largeur de 50, l'application effectuera les étapes suivantes :

1. Récupérez le 300 et le 50 des emplacements mémoire pour Canvas1.Width et ImageSprite.Width, respectivement.
2. Soustraie : $300 - 50 = 250$.
3. Appelez la fonction de fraction aléatoire pour obtenir un nombre compris entre 0 et 1 (par exemple, 0,5).
4. Multipliez : $250 * 0,5 = 125$.
5. Placez le 125 dans l'emplacement mémoire pour la propriété ImageSprite1.X .

Affichage des variables

Lorsque vous modifiez une propriété de composant, comme dans l'exemple précédent, l'interface utilisateur est directement affectée. Ce n'est pas vrai pour les variables ; changer une variable n'a aucun effet direct sur l'apparence de l'application. Si vous venez d'incrémenter un score variable sans modifier l'interface utilisateur d'une autre manière, l'utilisateur ne saura jamais qu'il y a eu un changement. C'est comme l'arbre proverbial qui tombe dans la forêt : si personne n'était là pour l'entendre, est-ce vraiment arrivé ?

Parfois, vous ne souhaitez pas manifester immédiatement une modification de l'interface utilisateur lorsqu'une variable change. Par exemple, dans un jeu, vous pouvez suivre des statistiques (par exemple, les tirs manqués) qui n'apparaîtront qu'à la fin du jeu.

C'est l'un des avantages du stockage des données dans une variable plutôt que dans une propriété de composant : vous pouvez afficher uniquement les données souhaitées lorsque vous souhaitez les afficher. Vous pouvez également séparer la partie informatique de votre application de l'interface utilisateur, ce qui facilite la modification ultérieure de cette interface utilisateur.

Par exemple, avec un jeu, vous pourriez stocker le score directement dans un Label ou dans une variable. Si vous le stockez dans une étiquette, vous incrémenterez la propriété Text de l'étiquette lorsque des points seront marqués, et l'utilisateur verra directement le changement. Si vous avez stocké le score dans une variable et incrémenté la variable lorsque des points ont été marqués, vous devrez inclure des blocs pour déplacer également la valeur de la variable dans une étiquette.

Cependant, si vous décidez de modifier l'application pour afficher la partition d'une manière différente, peut-être avec un curseur, la solution variable sera plus facile à modifier. Vous n'auriez pas besoin de modifier tous les endroits qui changent la partition ; il vous suffirait de modifier les blocs qui affichent le score.

Variables locales

Les variables décrites jusqu'à présent dans ce chapitre sont des variables globales et vous les définissez avec une initialisation globale à bloquer. Le « global » fait référence au fait que la variable peut être utilisée dans tous les gestionnaires d'événements et procédures. On dit que ces variables ont une portée globale.

Avec la dernière version d'App Inventor, vous pouvez désormais également définir des variables locales , c'est-à-dire des variables dont l'utilisation (portée) est limitée à un seul gestionnaire d'événements ou à une seule procédure (voir Figure 16-7).

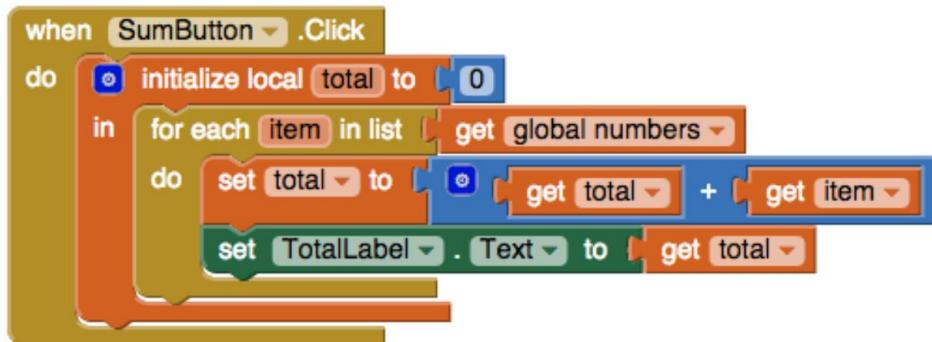


Figure 16-8. La variable « total » est locale ; il ne peut être utilisé que dans l'événement SumButton.Click

Si la variable n'est nécessaire qu'à un seul endroit, c'est une bonne idée de la définir comme locale, comme la variable « total » se trouve dans la figure 16-7. Ce faisant, vous limitez les dépendances dans votre application et vous assurez que vous ne modifiez pas par erreur une variable. Pensez à une variable locale comme la mémoire privée de votre cerveau : vous ne voulez certainement pas que d'autres cerveaux y aient accès !

Résumé

Lorsqu'une application est lancée, elle commence à exécuter ses opérations et à répondre aux événements qui se produisent. Lorsqu'elle répond à des événements, l'application doit parfois se souvenir de certaines choses.

Pour un jeu, il peut s'agir du score de chaque joueur ou de la direction dans laquelle un objet se déplace.

Votre application mémorise les éléments contenus dans les propriétés du composant, mais lorsque vous avez besoin d'emplacements de mémoire supplémentaires non associés à un composant, vous pouvez définir des variables. Vous pouvez stocker des valeurs dans une variable et récupérer la valeur actuelle, tout comme vous le faites avec les propriétés.

Comme pour les valeurs de propriété, les valeurs de variables ne sont pas visibles pour l'utilisateur final. Si vous souhaitez que l'utilisateur final voie les informations stockées dans une variable, vous ajoutez des blocs qui affichent ces informations dans une étiquette ou un autre composant de l'interface utilisateur.

Création d'applications animées

Ce chapitre traite des méthodes de création d'applications avec des animations simples (objets qui bougent).

Vous apprendrez les bases de la création de jeux bidimensionnels avec App Inventor et vous familiariserez avec les sprites d'images et la gestion d'événements tels que la collision de deux objets.

Lorsque vous voyez un objet se déplacer doucement sur l'écran de l'ordinateur, ce que vous voyez en réalité est une succession rapide d'images avec l'objet à chaque fois dans un endroit légèrement différent. C'est une illusion qui n'est pas très différente des flipbooks, dans lesquels vous voyez une image animée en feuilletant rapidement les pages. C'est le concept qui sous-tend la création des films d'animation !

Avec App Inventor, vous programmerez une animation en plaçant les composants Ball et ImageSprite dans un composant Canvas , puis en déplaçant et en transformant ces objets toutes les fractions de seconde successives. Dans ce chapitre, vous apprendrez comment fonctionne le système de coordonnées Canvas , comment utiliser l' événement Clock.Timer pour déclencher un mouvement, comment contrôler la vitesse des objets et comment réagir à des événements tels que la collision de deux objets. .

Ajout d'un composant Canvas à votre application

Depuis la palette Dessin et Animation, faites glisser un composant Canvas dans votre application.

Après l'avoir placé, spécifiez sa largeur et sa hauteur. Souvent, vous souhaiterez que le canevas s'étende sur toute la largeur de l'écran de l'appareil. Pour ce faire, choisissez « Remplir le parent » lors de la spécification de la largeur.

Vous pouvez faire la même chose pour la hauteur, mais en général, vous la définirez sur un certain nombre (par exemple, 300 pixels) pour laisser de la place à d'autres composants au-dessus et au-dessous du canevas.

Figure 17-1.



Le système de coordonnées du canevas

Un dessin sur une toile est en réalité une grille de pixels, où un pixel est le plus petit point de couleur possible pouvant apparaître à l'écran. L'emplacement de chaque pixel est défini par des coordonnées xy sur un système de grille, comme illustré sur la figure 17-1. Dans ce système de coordonnées, x définit un emplacement sur le plan horizontal (en commençant à 0 à l'extrême gauche et en augmentant à mesure que vous vous déplacez vers la droite sur l'écran), et y définit un emplacement sur le plan vertical (en commençant à 0 en haut) et augmentant à mesure que vous descendez l'écran).

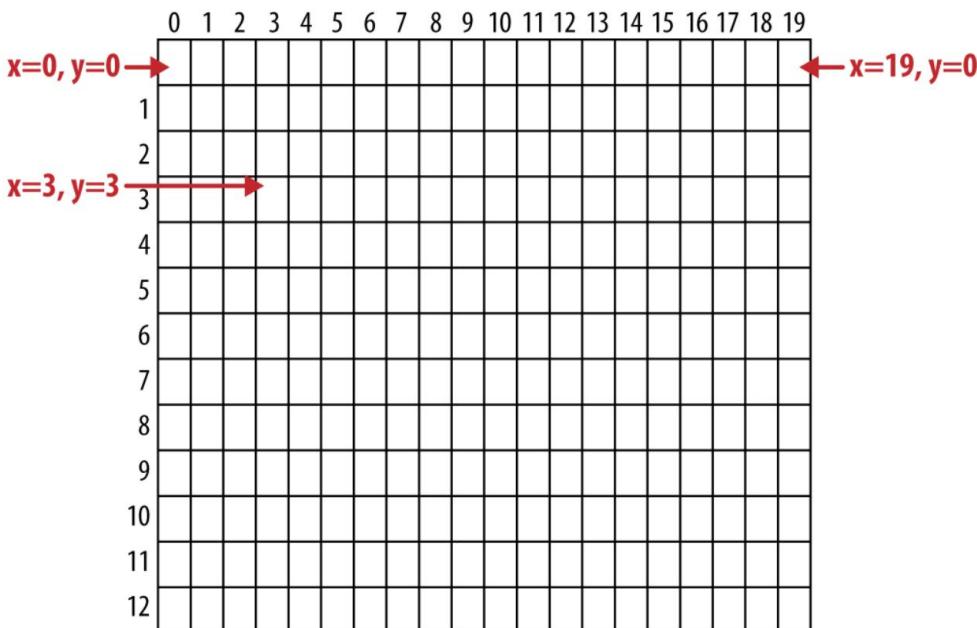


Figure 17-2. Le système de coordonnées Canvas

La cellule en haut à gauche d'un canevas commence par 0 pour les deux coordonnées, cette position est donc représentée par (x=0,y=0). À mesure que vous vous déplacez vers la droite, la coordonnée x augmente ; à mesure que vous descendez, la coordonnée y augmente. La cellule située immédiatement à droite du coin supérieur gauche est (x=1,y=0). Le coin supérieur droit a une coordonnée x égale à la largeur du canevas moins 1. La plupart des écrans de téléphone ont une largeur d'environ 300 pixels, mais pour l'exemple présenté ici, la largeur est de 20, donc le coin supérieur droit est la coordonnée. (x=19,y=0).

Vous pouvez modifier l'apparence de la toile de deux manières : en peignant dessus ou en y plaçant et en déplaçant des objets. Ce chapitre se concentre principalement sur ce dernier point, mais voyons d'abord comment « peindre » et comment créer une animation en peignant (c'est également le sujet de l'application PaintPot au chapitre 2).

Chaque cellule du Canvas contient un pixel définissant la couleur qui doit y apparaître. Le composant Canvas fournit les blocs Canvas.DrawLine et Canvas.DrawCircle pour peindre les pixels. Vous définissez d'abord la propriété Canvas.PaintColor sur la couleur souhaitée, puis appelez l'un des blocs Draw pour dessiner dans cette couleur. Avec DrawCircle, vous pouvez peindre des cercles de n'importe quel rayon, mais si vous définissez le rayon sur 1, comme le montre la figure 17-2, vous peindrez un pixel individuel.

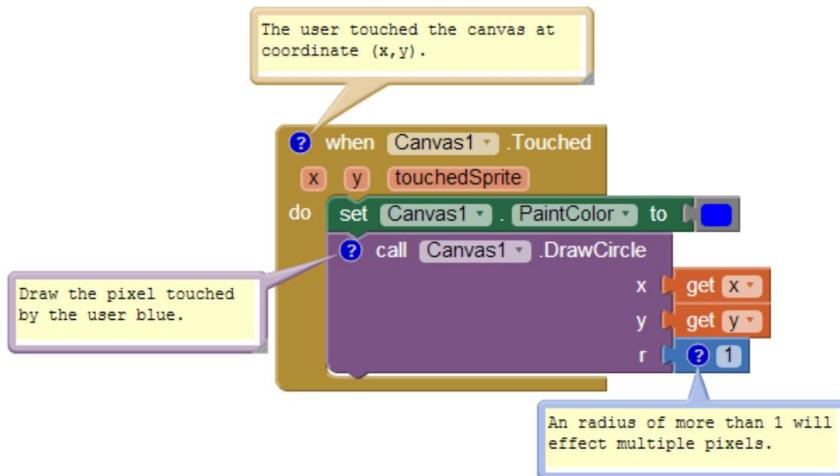


Figure 17-3. DrawCircle avec un rayon de 1 peint un pixel individuel à chaque touche

App Inventor fournit une palette de couleurs de base que vous pouvez utiliser pour peindre des pixels (ou d'autres composants de l'interface utilisateur). Vous pouvez accéder à une gamme plus large de couleurs en utilisant le schéma de numérotation des couleurs expliqué dans la documentation App Inventor à l'adresse <http://appinventor.mit.edu/explore/ai2/support/blocks/colors.html>.

En plus de peindre des pixels individuels, vous pouvez également placer des composants Ball et ImageSprite sur un canevas. Un sprite est un objet graphique placé dans une scène plus grande (dans App Inventor, la « scène » est un composant Canvas). Les composants Ball et ImageSprite sont des sprites ; ils ne diffèrent qu'en apparence. Une Balle est un cercle qui a une apparence qui ne peut être modifiée qu'en changeant sa couleur ou son rayon, alors qu'un ImageSprite peut prendre n'importe quelle apparence, telle que définie par un fichier image. Les boules et les ImageSprites ne peuvent être ajoutés que dans un canevas ; vous ne pouvez pas les faire glisser dans l'interface utilisateur en dehors de celle-ci.

Animation d'objets avec des événements de minuterie

Une façon de spécifier une animation dans App Inventor consiste à modifier un objet en réponse à un événement de minuterie. Le plus souvent, vous déplacerez les sprites vers différents emplacements sur le canevas à des intervalles de temps définis. L'utilisation d'événements de minuterie est la méthode la plus courante pour définir ces intervalles de temps définis. Plus tard, nous aborderons également une méthode alternative de programmation d'animation utilisant les propriétés Speed et Heading des composants ImageSprite et Ball .

Les clics sur les boutons et autres événements déclenchés par l'utilisateur sont simples à comprendre : l'utilisateur fait quelque chose et l'application répond en effectuant certaines opérations. Les événements de minuterie sont cependant différents, car ils ne sont pas déclenchés par l'utilisateur final mais plutôt par le passage du temps. Vous devez conceptualiser les événements déclenchant l'horloge du téléphone dans l'application au lieu qu'un utilisateur fasse quelque chose.

Pour définir un événement de minuterie, vous devez d'abord faire glisser un composant Clock dans votre application dans le Concepteur de composants. Le composant Clock est associé à une propriété TimerInterval . L'intervalle est défini en millisecondes (1/1 000 de seconde). Si vous définissez TimerInterval sur 500, cela signifie qu'un événement de minuterie sera déclenché toutes les demi-secondes. Plus le TimerInterval est petit, plus la fréquence d'images de l'animation est rapide.

Après avoir ajouté une horloge et défini un TimerInterval dans le concepteur, vous pouvez faire glisser un événement Clock.Timer dans l'éditeur de blocs. Vous pouvez placer tous les blocs de votre choix dans cet événement, et ils seront exécutés à chaque intervalle.

Créer du mouvement

Pour afficher un sprite se déplaçant dans le temps, vous utiliserez la fonction MoveTo trouvée dans les composants ImageSprite et Ball . Par exemple, pour déplacer une balle horizontalement sur l'écran, vous utiliserez des blocs comme ceux de la figure 17-3.

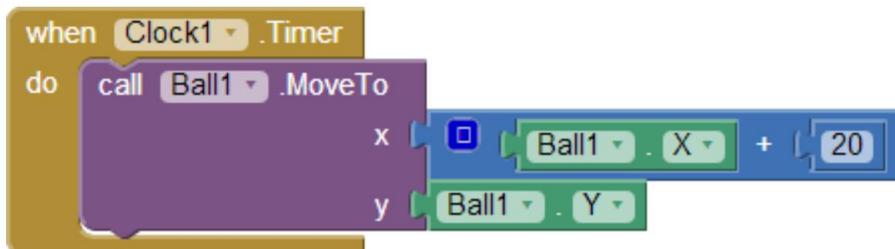


Figure 17-4. Déplacer la balle horizontalement sur l'écran

MoveTo déplace un objet vers un emplacement absolu sur le canevas, et non vers une valeur relative. Ainsi, pour déplacer un objet d'une certaine quantité, vous définissez les arguments MoveTo sur la valeur

l'emplacement actuel de l'objet plus un décalage. Parce que nous nous déplaçons horizontalement, l'argument x est défini sur l'emplacement x actuel (Ball1.X) plus le décalage 20, tandis que l'argument y est défini pour rester à sa valeur actuelle (Ball1.Y).

Pour déplacer un objet vers le bas, vous modifieriez uniquement les coordonnées Ball1.Y et laisseriez Ball1.X inchangé. Si vous souhaitez déplacer la balle en diagonale, vous ajouterez un décalage aux coordonnées x et y, comme le montre la figure 17-4.

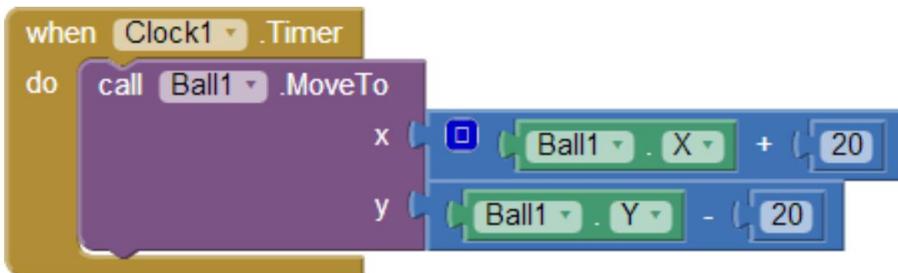


Figure 17-5. Décaler les coordonnées x et y pour déplacer la balle en diagonale

Vitesse

À quelle vitesse la balle se déplace-t-elle dans l'exemple précédent ? La vitesse dépend à la fois des paramètres que vous fournissez pour la propriété TimerInterval de Clock1 et des paramètres que vous spécifiez dans la fonction MoveTo . Si Clock.TimerInterval est défini sur 1 000 millisecondes, cela signifie qu'un événement Clock1.Timer sera déclenché toutes les secondes. Pour l'exemple horizontal illustré à la figure 17-3, la balle se déplacera à une vitesse de 20 pixels par seconde.

Mais un TimerInterval de 1 000 millisecondes ne fournit pas une animation très fluide ; la balle ne bougera qu'une fois par seconde, ce qui semblera saccadé. Pour obtenir un mouvement plus fluide, vous avez besoin d'un intervalle plus court. Si le TimerInterval était plutôt réglé sur 100 millisecondes, la balle se déplacerait de 20 pixels tous les dixièmes de seconde, soit 200 pixels par seconde, une vitesse qui semblerait beaucoup plus fluide.

Détection de collision

Pour créer des jeux et autres applications animées, vous avez besoin de fonctionnalités plus complexes que le simple mouvement. Heureusement, App Inventor fournit des blocs de haut niveau pour gérer les événements d'animation tels qu'un objet atteignant le bord de l'écran ou la collision de deux objets.

Dans ce contexte, le blocage de haut niveau signifie qu'App Inventor s'occupe des détails de niveau inférieur pour déterminer le moment où un événement tel qu'une collision se produit. Vous pouvez vérifier ces occurrences manuellement en vérifiant les propriétés des sprites et des canevas dans les événements Clock.Timer . Ce niveau de programmation nécessite une logique assez complexe,

cependant. Heureusement, App Inventor les fournit pour vous, et c'est une bonne chose car ces événements sont communs à de nombreux jeux et autres applications.

Bord atteint

Considérons à nouveau l'animation de la figure 17-4, dans laquelle l'objet se déplace en diagonale du coin supérieur gauche au coin inférieur droit de la toile. Comme programmé, la balle se déplacerait en diagonale, puis s'arrêterait lorsqu'elle atteignait le bord droit ou inférieur de la toile (le système ne déplacerait pas un objet au-delà des limites de la toile).

Si vous souhaitiez plutôt que l'objet réapparaisse dans le coin supérieur gauche à chaque fois qu'il atteint le bord inférieur ou droit, vous pouvez définir une réponse à l'événement Ball.EdgeReached similaire à celle illustrée dans la figure 17-5.



Figure 17-6. Faire réapparaître la balle dans le coin supérieur gauche lorsqu'elle atteint un bord

EdgeReached est déclenché lorsqu'une balle ou un sprite d'image frappe n'importe quel bord d'une toile. Ce gestionnaire d'événement, combiné au mouvement diagonal spécifié avec l'événement de minuterie décrit précédemment (Figure 17-4), fait que la balle se déplace en diagonale du coin supérieur gauche vers le coin inférieur droit, remonte vers le coin supérieur gauche lorsqu'elle atteint un bord, et puis recommencez, pour toujours (ou jusqu'à ce que l'application vous dise le contraire).

Pour cet exemple, nous n'avons pas fait de distinction entre les bords touchés. L'événement EdgeReached possède un paramètre, edge, qui spécifie le bord particulier à l'aide du code suivant :

- Nord = 1
- Nord-est = 2
- Est = 3
- Sud-est = 4
- Sud = -1
- Sud-ouest = -2

- Ouest = -3
- Nord-Ouest = -4

CollidingWith et NoLongerCollidingWith

Les jeux et autres applications animées reposent souvent sur une activité se produisant lorsque deux objets ou plus entrent en collision (par exemple, une batte frappant une balle).

Prenons par exemple un jeu dans lequel un objet change de couleur et émet un son d'explosion lorsqu'il heurte un autre objet. La figure 17-6 montre les blocs d'un tel gestionnaire d'événements.

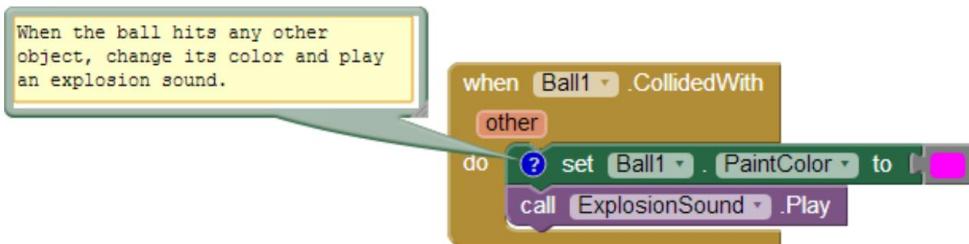


Figure 17-7. Faire changer la couleur de la balle et émettre un son d'explosion lorsqu'elle touche un autre objet

NoLongerCollidingWith fournit l'événement opposé de CollidedWith. Il se déclenche uniquement lorsque deux objets se sont rapprochés puis se sont séparés. Ainsi, pour votre jeu, vous pouvez inclure les blocs représentés dans la figure 17-7.



Figure 17-8. Changer la couleur et arrêter le bruit d'explosion lorsque les objets se séparent

Notez que CollidedWith et NoLongerCollidingWith ont tous deux un argument, autre, qui spécifie l'objet particulier avec lequel vous êtes entré en collision (ou dont vous vous êtes séparé). Cela vous permet d'effectuer des opérations uniquement lorsque l'objet (par exemple, Ball1) interagit avec un autre objet particulier, comme le montre la figure 17-8.

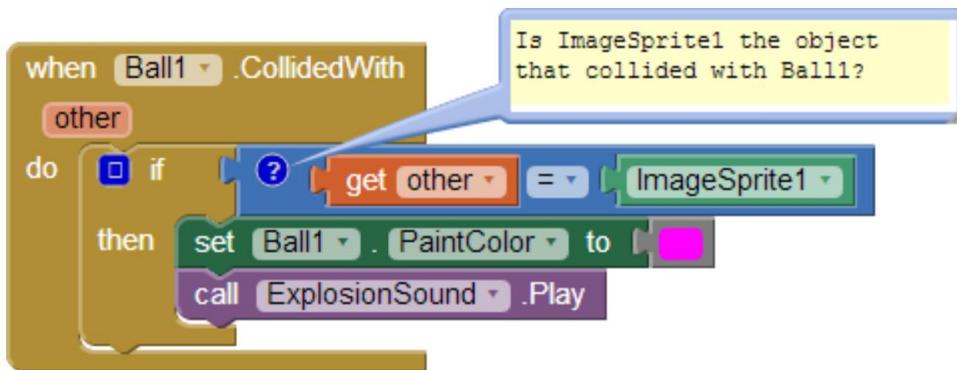


Figure 17-9. Effectuez la réponse uniquement si Ball1 frappe ImageSprite1

Le bloc **ImageSprite1** en est un dont nous n'avons pas encore discuté. Ce bloc fait référence au composant dans son ensemble et non à une propriété particulière du composant. Lorsque vous avez besoin de comparer des composants (par exemple pour savoir lesquels sont entrés en collision), vous utilisez ce bloc. Chaque composant possède un tel bloc dans son tiroir, et le bloc porte le même nom que le composant.

Animation interactive

Dans les comportements animés dont nous avons parlé jusqu'à présent, l'utilisateur final n'est pas impliqué. Les jeux sont bien entendu interactifs, l'utilisateur final jouant un rôle central. Souvent, l'utilisateur final contrôle la vitesse ou la direction d'un objet avec des boutons ou d'autres objets de l'interface utilisateur.

A titre d'exemple, mettons à jour l'animation diagonale en donnant à l'utilisateur la possibilité de arrêtez et démarrez le mouvement diagonal. Vous pouvez le faire en programmant un gestionnaire d'événements **Button.Click** pour désactiver et réactiver l'événement de minuterie de l'horloge. composant.

Par défaut, la propriété **timerEnabled** du composant **Clock** est cochée. Vous pouvez le désactiver dynamiquement en le définissant sur **false** dans un gestionnaire d'événements. Le gestionnaire d'événements de la figure 17-9, par exemple, arrêterait l'activité d'un minuteur d'horloge au premier clic.



Figure 17-10. Arrêter le chronomètre la première fois que l'on clique sur le bouton

Une fois que la propriété **Clock1.TimerEnabled** est définie sur **false**, l'événement **Clock1.Timer** ne se déclenchera plus et la balle cessera de bouger.

Bien sûr, arrêter le mouvement au premier clic n'est pas très intéressant. Plutôt, vous pouvez « basculer » le mouvement de la balle en ajoutant un if else dans le gestionnaire d'événements qui active ou désactive le chronomètre, comme le montre la figure 17-10.

Ce gestionnaire d'événements arrête le minuteur au premier clic et réinitialise le bouton pour qu'il affiche « Démarrer » au lieu de « Stop ». La deuxième fois que l'utilisateur clique sur le bouton, TimerEnabled est faux, donc la partie « else » est exécutée. Dans ce cas, la minuterie est activée, ce qui fait bouger à nouveau l'objet, et le texte du bouton revient à « Stop ».

Pour plus d'informations sur les blocs ifelse , consultez le chapitre 18 et pour des exemples d'animations interactives utilisant le capteur d'orientation, consultez le chapitre 5 et le chapitre 23.

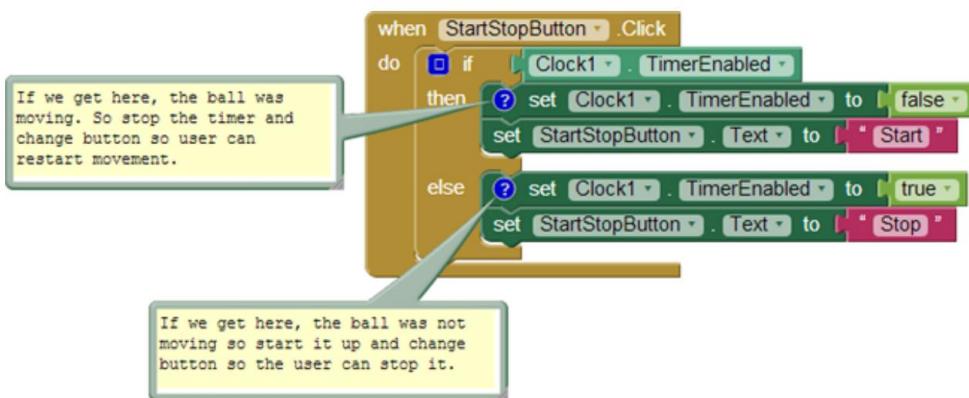


Figure 17-11. Ajout d'un if else pour que cliquer sur le bouton démarre et arrête le mouvement de la balle

Spécification d'une animation de sprite sans minuterie d'horloge

Les exemples d'animation décrits jusqu'à présent utilisent un composant Clock et spécifient qu'un objet doit se déplacer à chaque fois que l'événement Clock.Timer est déclenché. Le schéma d'événements Clock.Timer est la méthode la plus générale pour spécifier une animation. Au-delà du simple déplacement d'un objet, vous pouvez également lui faire changer la couleur d'un objet au fil du temps, modifier du texte (pour donner l'impression que l'application est en train de taper) ou demander à l'application de prononcer des mots à un certain rythme.

Si vous souhaitez uniquement déplacer des objets, App Inventor propose une alternative qui ne le fait pas. nécessitent l'utilisation d'un composant Clock . Comme vous l'avez peut-être remarqué, les composants ImageSprite et Ball ont des propriétés de titre, de vitesse et d'intervalle. Au lieu de définir un gestionnaire d'événements Clock.Timer , vous pouvez définir ces propriétés dans le concepteur de composants ou l'éditeur de blocs pour contrôler la façon dont un sprite se déplace.

Pour illustrer, reconsidérons l'exemple qui a déplacé une balle en diagonale. Le titre La propriété d'un sprite ou d'une balle a une portée de 360 degrés, comme illustré dans la figure 17-11.

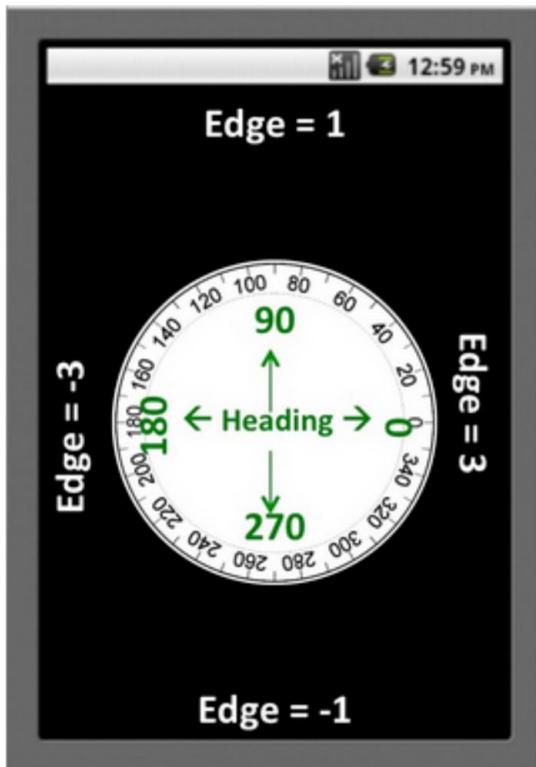


Figure 17-12. La propriété Heading a une plage de 360 degrés

Si vous réglez le cap sur 0, la balle se déplacera de gauche à droite. Si vous le réglez sur 90, il se déplacera de bas en haut. Si vous le réglez sur 180, il se déplacera de droite à gauche. Si vous le réglez sur 270, il se déplacera de haut en bas. Et si vous le réglez sur 315, la balle se déplacera du coin supérieur gauche vers le coin inférieur droit.

Pour déplacer un objet, vous devez également définir la propriété Vitesse sur une valeur autre que 0. La vitesse à laquelle l'objet se déplace est en fait déterminée par les propriétés Vitesse et Intervalle ensemble. La propriété Vitesse est la distance, en pixels, sur laquelle l'objet se déplacera à chaque intervalle.

Pour tester ces propriétés, créez une application de test avec un Canvas et une Ball et connectez-vous votre appareil ou émulateur pour des tests en direct. Ensuite, modifiez les propriétés de cap, de vitesse et d'intervalle de la balle pour voir comment elles fonctionnent.

Par exemple, supposons que vous vouliez déplacer une balle d'avant en arrière du coin supérieur gauche au coin inférieur droit de la toile. Dans le Designer, vous pouvez initialiser la vitesse de la balle à 5 et l'intervalle à 100, puis définir la propriété Heading sur 315. Vous ajouterez ensuite le gestionnaire d'événements Ball1.EdgeReached , que vous pouvez voir dans la figure 17-12, pour modifier la direction de la balle lorsqu'elle atteint l'un ou l'autre bord.

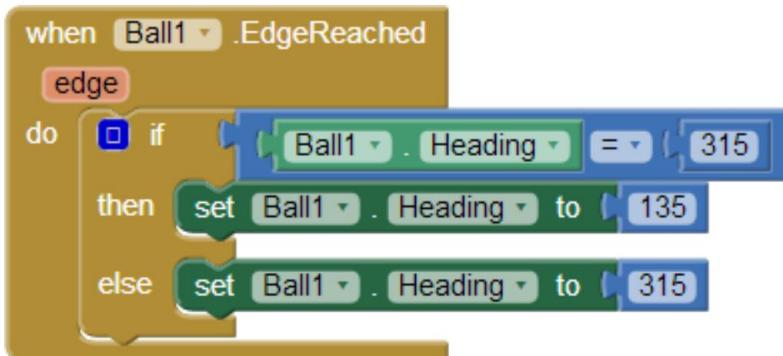


Figure 17-13. Changer la direction de la balle lorsqu'elle atteint l'un ou l'autre bord

Résumé

À l'aide du composant Canvas , vous pouvez définir une sous-zone de l'écran de l'appareil dans laquelle les objets peuvent se déplacer et interagir. Vous ne pouvez placer que deux types de composants dans un canevas : les ImageSprites et les Balls.

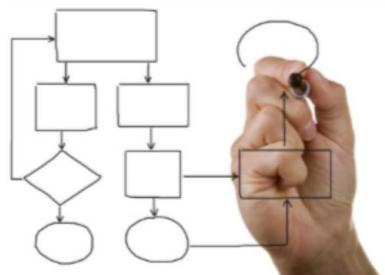
L'animation est un objet en mouvement ou en transformation au fil du temps. Vous pouvez programmer une animation, y compris des mouvements et d'autres transformations graphiques, avec l'événement Timer du composant Clock . Si vous souhaitez simplement déplacer des objets, vous pouvez utiliser une méthode alternative basée sur les propriétés Heading, Speed et Interval internes aux composants ImageSprite et Ball .

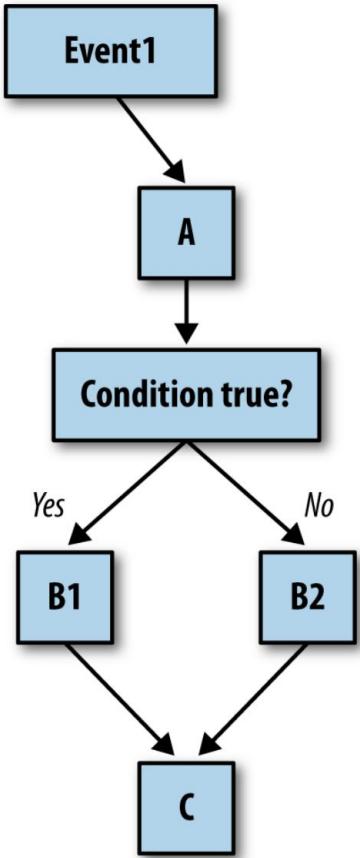
Quelle que soit la méthode, vous pouvez également profiter de fonctionnalités de haut niveau pour gérer les événements qui gèrent les collisions.

Programmer votre application pour prendre des décisions : Blocs conditionnels

Les ordinateurs, même les plus petits comme le téléphone dans votre poche, sont capables d'effectuer des millions d'opérations en une seule seconde. Plus impressionnant encore, ils peuvent également prendre des décisions basées sur les données de leurs banques de mémoire et la logique spécifiée par le programmeur. Cette capacité de prise de décision est probablement l'ingrédient clé de ce que les gens considèrent comme l'intelligence artificielle, et c'est certainement un élément très important dans la création d'applications intelligentes et intéressantes ! Dans ce chapitre, nous explorerons comment intégrer cette logique de prise de décision dans vos applications.

Figure 18-1.





Le chapitre 14 explique comment le comportement d'une application est défini par un ensemble de gestionnaires d'événements. Chaque gestionnaire d'événements exécute des fonctions spécifiques en réponse à un événement particulier. Toutefois, la réponse ne doit pas nécessairement être une séquence linéaire de fonctions ; vous pouvez spécifier que certaines fonctions ne soient exécutées que sous certaines conditions. Par exemple, une application de jeu peut vérifier si le score d'un joueur a atteint 100, ou une application de localisation peut demander si le téléphone se trouve dans les limites d'un bâtiment. Votre application peut poser de telles questions et, en fonction de la réponse, procéder en conséquence.

Considérez le diagramme de la figure 18-1. Quand le événement se produit, la fonction (bloc) A est exécutée. Ensuite, un test de décision est effectué. Si le test est vrai, B1 est effectué. Si c'est faux, B2 est exécuté. Dans les deux cas, le reste du gestionnaire d'événements (C) est terminé.

Étant donné que les diagrammes de décision d'application comme celui-ci ressemblent à des arbres, nous disons que l'application « se branche » dans un sens ou dans l'autre en fonction du résultat du test. Ainsi, dans ce cas, vous diriez : « Si le test est vrai, la branche contenant B1 est effectuée. »

Figure 18-2. Un gestionnaire d'événements qui teste une condition et se branche en conséquence

Conditions de test avec if et else if blocs

Pour autoriser le branchement conditionnel, App Inventor fournit un bloc conditionnel if-then dans le tiroir Contrôle. Vous pouvez étendre le bloc avec autant de branches else et else if que vous le souhaitez en cliquant sur l'icône bleue, comme le montre la figure 18-2.

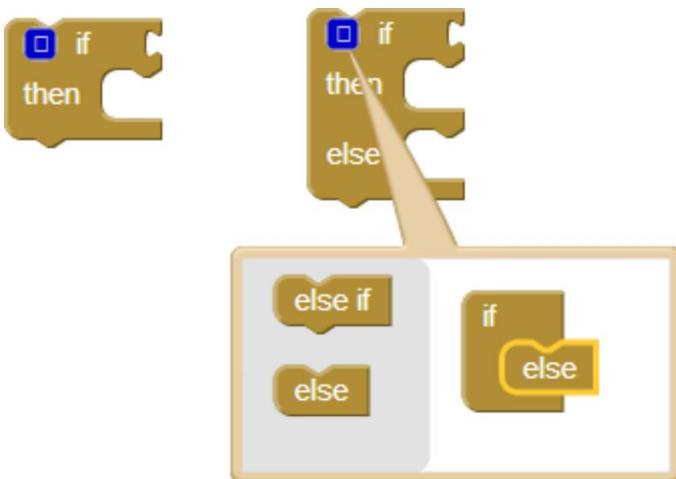


Figure 18-3. Les blocs conditionnels if et else if

Vous pouvez brancher n'importe quelle expression booléenne dans les sockets de test des blocs if et else if . Une expression booléenne est une équation mathématique qui renvoie un résultat vrai ou faux. L'expression teste la valeur des propriétés et des variables à l'aide d'opérateurs relationnels et logiques tels que ceux illustrés dans la figure 18-3.

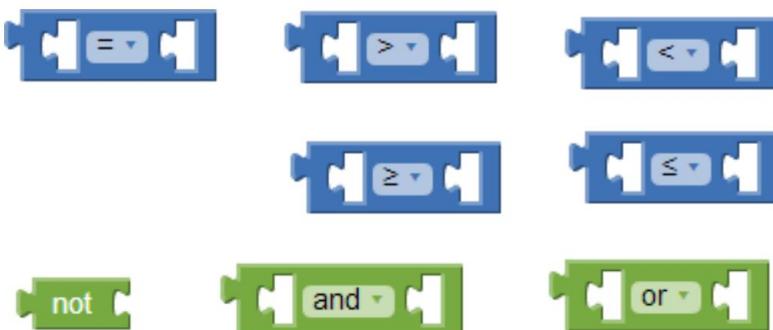


Figure 18-4. Blocs d'opérateurs relationnels et logiques utilisés dans les tests conditionnels

Les blocs que vous placez dans le socket « then » d'un bloc if ne seront exécutés que si le test est vrai. Si le test est faux, l'application passe aux blocs suivants.

Pour un jeu, vous pouvez insérer une expression booléenne pour vérifier le score d'un joueur, comme le montre la figure 18-4.



Figure 18-5. Une expression booléenne utilisée pour tester la valeur de la variable score

Dans cet exemple, un fichier sonore est joué si le score est supérieur à 100. Dans cet exemple, si le test est faux, le son n'est pas joué et l'application passe en dessous de la totalité du bloc if-then et passe au suivant. bloquer dans votre application. Si vous souhaitez qu'un faux test déclenche une action, vous pouvez utiliser un bloc else ou else if .

Programmation d'une décision Soit/Ou

Pensez à une application que vous pourriez utiliser lorsque vous vous ennuyez : vous appuyez sur un bouton de votre téléphone et elle appelle un ami au hasard. Dans la figure 18-5, un bloc entier aléatoire est utilisé pour générer un nombre aléatoire, puis un bloc if else appelle un numéro de téléphone particulier en fonction de ce nombre aléatoire.

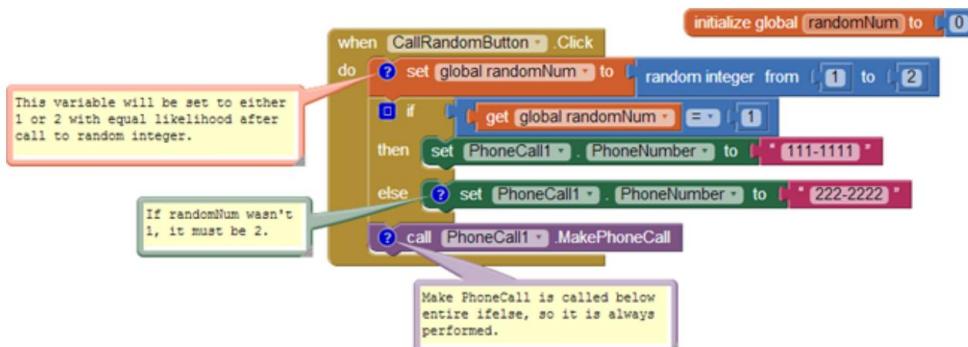


Figure 18-6. Sinon, si le bloc appelle l'un des deux nombres en fonction de l'entier généré aléatoirement

Dans cet exemple, un entier aléatoire est appelé avec les arguments 1 et 2, ce qui signifie que le nombre aléatoire renvoyé sera 1 ou 2 avec une probabilité égale. La variable RandomNum stocke le nombre aléatoire renvoyé.

Après avoir défini RandomNum, les blocs le comparent au numéro 1 dans le test if . Si la valeur de RandomNum est 1, l'application prend la première branche (puis) et le numéro de téléphone est défini sur 111-1111. Si la valeur n'est pas 1, alors le test est faux, auquel cas l'application prend la deuxième branche (sinon) et le numéro de téléphone est défini sur 222-2222. L'application fait

l'appel téléphonique dans tous les cas, car l'appel à MakePhoneCall est inférieur à la totalité du bloc if else .

Conditions de programmation dans les conditions

De nombreuses situations de décision comportent plus de deux résultats parmi lesquels choisir. Par exemple, vous souhaiterez peut-être choisir entre plus de deux amis dans votre programme d'appel aléatoire. Pour ce faire, vous pouvez placer une branche else if avant la branche else d'origine , comme le montre la figure 18-6.

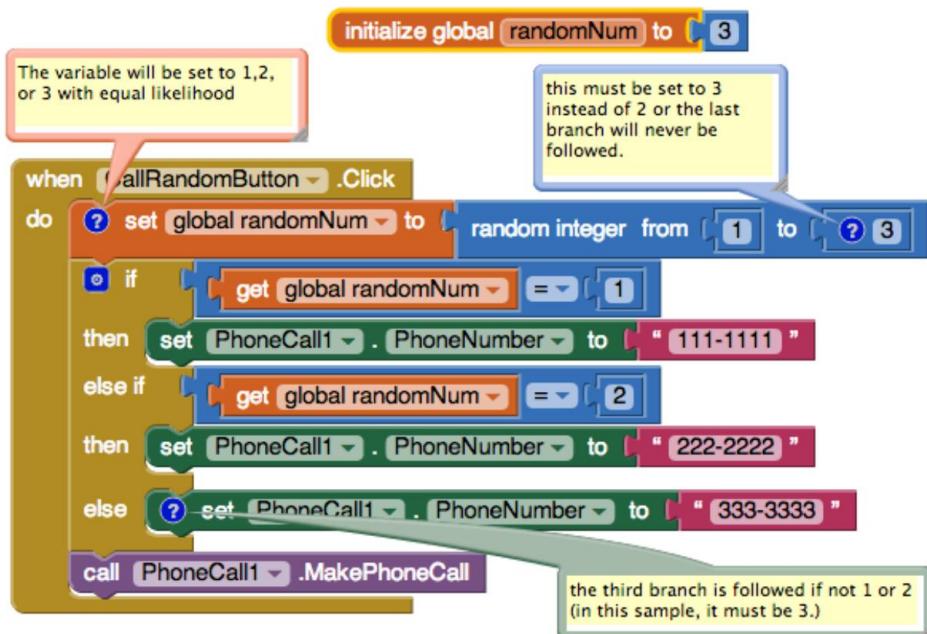


Figure 18-7. if, else if et else fournir trois branches possibles

Avec ces blocs, si le premier test est vrai, l'application exécute la première branche then-do et appelle le numéro 111-1111. Si le premier test est faux, la branche else if est exécutée, ce qui exécute immédiatement un autre test. Ainsi, si le premier test (RandomNum=1) est faux et que le second (RandomNum=2) est vrai, la deuxième branche est exécutée et 222-2222 est appelée. Si les deux tests sont faux, sinon la branche en bas est exécutée et le troisième numéro (333-3333) est appelé.

Notez que cette modification ne fonctionne que parce que le paramètre to de l' appel d'entier aléatoire a été modifié à 3 afin que 1, 2 ou 3 soit généré avec une probabilité égale.

Vous pouvez ajouter autant de branches if que vous le souhaitez. Vous pouvez également imbriquer des conditions dans les conditions. Lorsque des tests conditionnels sont placés dans des branches d'un autre test conditionnel, on dit qu'ils sont imbriqués. Vous pouvez imbriquer des conditions et d'autres constructions de contrôle telles que des boucles pour chaque à des niveaux arbitraires afin d'ajouter de la complexité à votre application.

Programmation de conditions complexes

Outre les conditions d'imbrication, vous pouvez également spécifier des tests conditionnels uniques qui sont plus complexes qu'un simple test d'égalité. Par exemple, considérons une application qui vibre lorsque votre téléphone (et probablement vous !) quitte un bâtiment ou une limite. Une telle application pourrait être utilisée par une personne en probation pour l'avertir lorsqu'elle s'éloigne trop de ses limites légales. Les parents peuvent l'utiliser pour surveiller où se trouvent leurs enfants. Un enseignant peut l'utiliser pour prendre automatiquement part (si tous ses élèves ont un téléphone Android !).

Pour cet exemple, posons cette question : le téléphone se trouve-t-il dans les limites du Harvey Science Center de l'Université de San Francisco ? Une telle application nécessiterait un test complexe composé de quatre questions différentes :

- La latitude du téléphone est-elle inférieure à la latitude maximale (37,78034) du frontière ?
- La longitude du téléphone est-elle inférieure à la longitude maximale (-122,45027) du frontière ?
- La latitude du téléphone est-elle supérieure à la latitude minimale (37,78016) du frontière ?
- La longitude du téléphone est-elle supérieure à la longitude minimale (-122,45059) du frontière ?

Vous avez besoin du composant LocationSensor pour cet exemple. Vous devriez pouvoir suivre ici même si vous n'avez pas été exposé à LocationSensor, mais vous pouvez en apprendre davantage à ce sujet au chapitre 23.

Vous pouvez créer des tests complexes en utilisant les opérateurs logiques et, ou, et non, que vous pouvez trouver dans le tiroir Logique. Dans ce cas, vous faites glisser un bloc if et quelques blocs and , placez l'un des blocs and dans le socket « test » du if, et les autres dans le premier bloc and , comme illustré dans la figure 18-7.

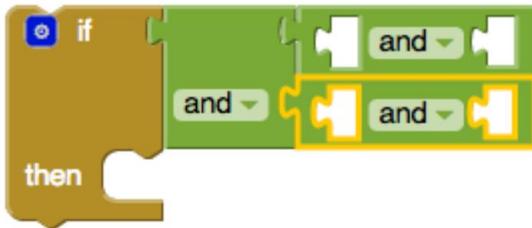


Figure 18-8. Un test if peut tester de nombreuses conditions en utilisant et, ou, et d'autres blocs relationnels

Vous devez ensuite faire glisser les blocs pour la première question et les placer dans le socket « test » du premier bloc, comme le montre la figure 18-8.



Figure 18-9. Les blocs du premier test sont placés dans le bloc et

Vous pouvez ensuite remplir les autres sockets avec les autres tests et placer l'intégralité du if dans un événement LocationSensor.LocationChanged . Vous disposez désormais d'un gestionnaire d'événements qui vérifie la limite, comme illustré dans la figure 18-9.

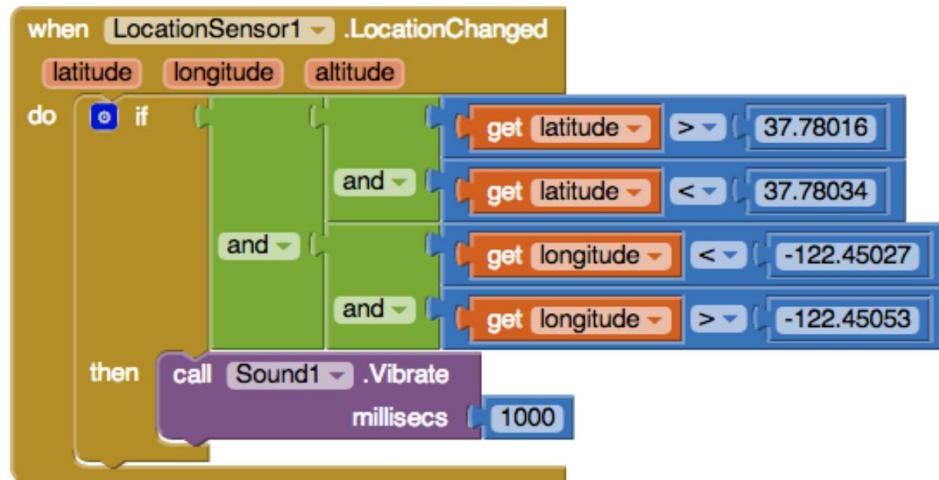


Figure 18-10. Ce gestionnaire d'événements vérifie la limite à chaque fois que l'emplacement change

Avec ces blocs, chaque fois que le LocationSensor obtient une nouvelle lecture et son emplacement est à l'intérieur des limites, le téléphone vibre.

OK, jusqu'à présent, c'est plutôt cool, mais essayons maintenant quelque chose d'encore plus compliqué à vous donner une idée de toute l'étendue des pouvoirs de décision de l'application. Et si vous vouliez que le téléphone vibre uniquement lorsque la frontière est franchie de l'intérieur vers l'extérieur ? Avant d'aller de l'avant, réfléchissez à la manière dont vous pourriez programmer une telle condition.

Notre solution consiste à définir une variable à l'intérieur de la limite qui se souvient si la lecture précédente du capteur se trouvait à l'intérieur ou à l'extérieur de la limite, puis à la comparer à chaque lecture successive du capteur. `insideBoundary` est un exemple de variable booléenne : au lieu de stocker un nombre ou un texte, elle stocke vrai ou faux. Pour cet exemple, vous l'initialiserez sur false, comme le montre la figure 18-11, ce qui signifie que l'appareil ne se trouve pas au sein du Harney Science Center de l'USF.

```
initialize global [insideBoundary] to [false]
```

Figure 18-11. `insideBoundary` est initialisé à false

Les blocs peuvent maintenant être modifiés pour que la variable `insideBoundary` soit définie sur chaque bloc. changement d'emplacement, et pour que le téléphone vibre uniquement lorsqu'il se déplace de l'intérieur vers l'extérieur de la limite. Pour dire cela en termes que nous pouvons utiliser pour les blocs, le téléphone doit vibrer lorsque 1) la variable `insideBoundary` est vraie, ce qui signifie que la lecture précédente était à l'intérieur de la limite, et 2) la nouvelle lecture du capteur de localisation est en dehors de la limite. La figure 18-11 montre les blocs mis à jour.

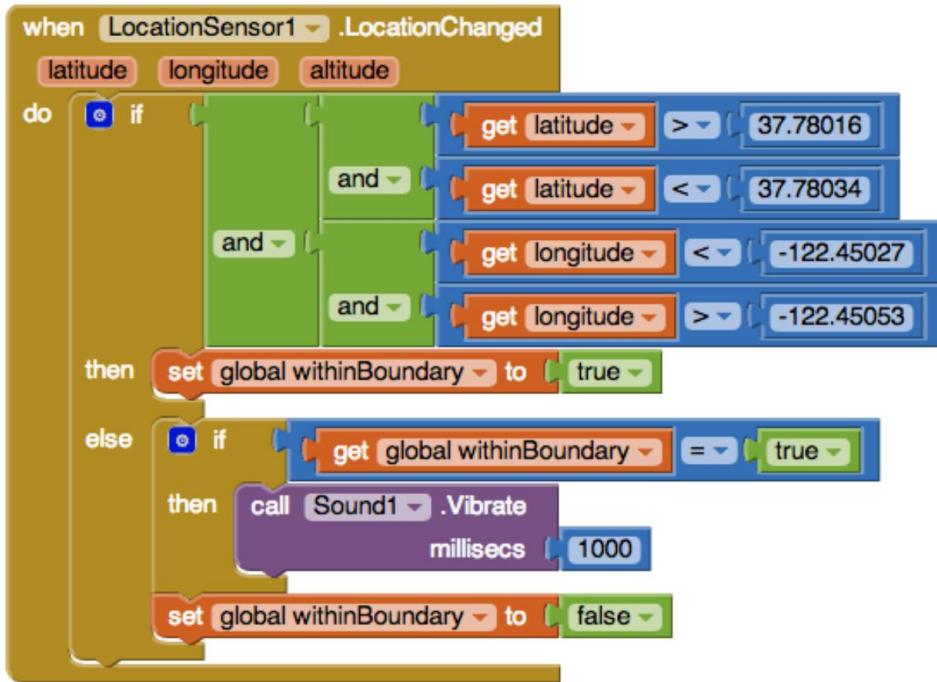


Figure 18-12. Ces blocs font vibrer le téléphone uniquement lorsqu'il se déplace de l'intérieur de la limite vers l'extérieur de la limite.

Examinons ces blocs de plus près. Lorsque le LocationSensor obtient une lecture, il vérifie d'abord si la nouvelle lecture se situe dans la limite. Si tel est le cas, LocationSensor définit la variable insideBoundary sur true. Parce que nous voulons que le téléphone vibre uniquement lorsque nous sommes en dehors de la limite, aucune vibration n'a lieu dans cette première branche.

Si nous arrivons à l'autre, nous savons que la nouvelle lecture se situe en dehors des limites. À ce stade, nous devons vérifier la lecture précédente : si nous sommes en dehors de la limite, nous voulons que le téléphone vibre uniquement si la lecture précédente était à l'intérieur de la limite. insideBoundary nous donne la lecture précédente, afin que nous puissions le vérifier. Si c'est vrai, on fait vibrer le téléphone.

Il y a encore une chose que nous devons faire après avoir vérifié que le téléphone a déplacé de l'intérieur vers l'extérieur de la frontière – pouvez-vous penser à ce que c'est ? Nous devons également réinitialiser WithinBoundary sur false afin que le téléphone ne vibre plus lors de la prochaine lecture du capteur.

Une dernière remarque sur les variables booléennes : consultez les deux tests if de la figure 18-12. Sont-ils équivalents ?



Figure 18-13. Pouvez-vous dire si ces deux tests sont équivalents ?

La réponse est oui!" La seule différence est que le test de droite est en réalité le manière plus sophistiquée de poser la question. Le test de gauche compare la valeur d'une variable booléenne avec vrai. Si WithinBoundary contient vrai, vous comparez vrai à vrai, ce qui est vrai. Si la variable contient faux, vous comparez faux à vrai, ce qui est faux. Cependant, le simple fait de tester la valeur de insideBoundary, comme dans le test de droite, donne le même résultat et est plus facile à coder.

Résumé

Votre tête tourne ? Ce dernier comportement était assez complexe ! Mais c'est le type de prise de décision que les applications sophistiquées doivent effectuer. Si vous construisez de tels comportements partie par partie (ou branche par branche) et testez au fur et à mesure, vous constaterez que spécifier une logique complexe – voire, oserons-nous dire, une intelligence artificielle – est réalisable. Cela vous fera mal à la tête et exercera un peu le côté logique de votre cerveau, mais cela peut aussi être très amusant.

Programmation des listes de données

Comme vous l'avez déjà vu, les applications gèrent les événements et prennent des décisions ; un tel traitement est fondamental en informatique. Mais l'autre élément fondamental d'une application réside dans ses données, c'est-à-dire les informations qu'elle traite. Les données d'une application sont rarement limitées à un seul emplacement mémoire, comme le score d'un jeu. Le plus souvent, il s'agit de listes d'informations et d'éléments complexes et interdépendants qui doivent être organisés avec autant de soin que les fonctionnalités de l'application.

Dans ce chapitre, nous examinerons la manière dont App Inventor gère les données. Vous apprendrez les principes fondamentaux de la programmation à la fois des informations statiques, dans lesquelles les données ne changent pas, et des informations dynamiques, dans lesquelles les données sont saisies par l'utilisateur final. Vous apprendrez à travailler avec des listes, puis vous explorerez une structure de données plus complexe impliquant des listes de listes et une application de quiz à choix multiples.

De nombreuses applications traitent des listes de données. Par exemple, Facebook traite votre liste de amis et listes de rapports de statut. Une application de quiz fonctionne avec une liste de questions et de réponses. Un jeu peut avoir une liste de personnages ou des scores records.

Vous spécifiez des données de liste dans App Inventor avec une variable, mais au lieu de nommer une seule cellule mémoire avec la variable, vous nommez un ensemble de cellules mémoire associées. Vous spécifiez qu'une variable est multi-éléments en utilisant soit la création d'une liste , soit la création de blocs de liste vides . Par exemple, la variable phoneNumbers dans la figure 19-1 définit une liste de trois éléments.



Figure 19-1.

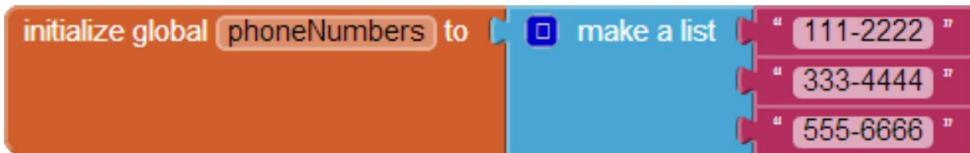


Figure 19-2. phoneNumbers nomme trois cellules mémoire initialisées avec les numéros affichés

Création d'une variable de liste

Vous créez une variable de liste dans l'éditeur de blocs en utilisant un bloc de variable globale d'initialisation , puis en connectant un bloc de création de liste . Vous pouvez trouver le bloc Créer une liste dans le tiroir Listes, et il n'a que deux sockets. Mais vous pouvez spécifier le nombre de sockets souhaités dans la liste en cliquant sur l'icône bleue et en ajoutant des éléments, comme illustré dans la figure 19-2.

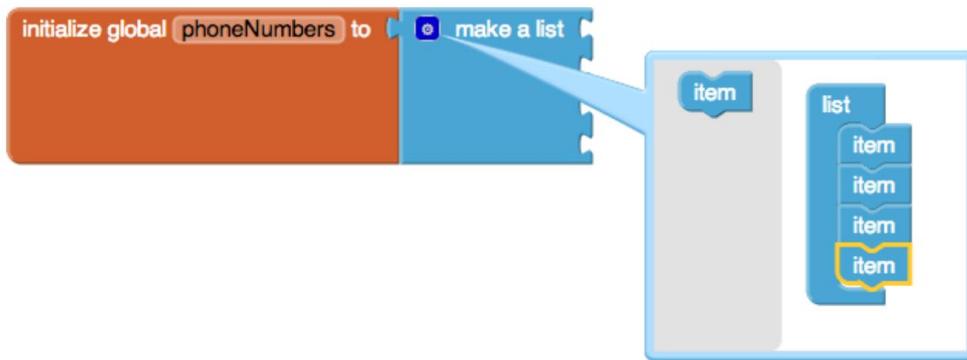


Figure 19-3. Cliquez sur l'icône bleue pour créer une liste pour modifier le nombre d'éléments

Vous pouvez brancher n'importe quel type de données dans les prises « élément » pour créer une liste. Pour l'exemple phoneNumbers, les éléments doivent être des objets texte, pas des nombres, car les numéros de téléphone comportent des tirets et d'autres symboles de formatage que vous ne pouvez pas mettre dans un objet numérique, et vous n'effectuerez aucun calcul sur les nombres (dans lesquels Dans ce cas, vous voudriez plutôt numéroté des objets).

Sélection d'un élément dans une liste

Pendant l'exécution de votre application, vous devrez sélectionner des éléments dans la liste ; par exemple, une question particulière lorsque l'utilisateur parcourt un quiz ou un numéro de téléphone particulier choisi dans une liste. Vous accédez aux éléments d'une liste à l'aide d'un index ; c'est-à-dire en spécifiant une position dans la liste. Si une liste comporte trois éléments, vous pouvez accéder aux éléments en utilisant les index 1, 2 et 3. Vous pouvez utiliser le bloc de sélection d'élément de liste pour récupérer un élément particulier, comme le montre la figure 19-3.



Figure 19-4. Sélection du deuxième élément d'une liste

Avec select list item, vous branchez la liste souhaitée dans la première socket et l'index souhaité dans la deuxième socket. Pour cet exemple de numéro de téléphone, le résultat de la sélection du deuxième élément est « 333-4444 ».

Utiliser un index pour parcourir une liste

Dans de nombreuses applications, vous définirez une liste de données, puis permettrez à l'utilisateur de la parcourir (ou de la parcourir). Le Quiz des présidents du chapitre 8 en fournit un bon exemple : dans cette application, lorsque l'utilisateur clique sur un bouton Suivant, l'élément suivant est sélectionné dans une liste de questions et affiché.

La section précédente a montré comment sélectionner le deuxième élément d'une liste, mais comment sélectionner l'élément suivant ? Lorsque vous parcourez une liste, le numéro d'élément que vous sélectionnez change à chaque fois ; c'est votre position actuelle dans la liste. Par conséquent, vous devez définir une variable pour représenter cette position actuelle. « index » est le nom commun d'une telle variable, et elle est généralement initialisée à 1 (la première position dans la liste), comme le montre la figure 19-4.



Figure 19-5. Initialisation de l'index de la variable à 1

Lorsque l'utilisateur fait quelque chose pour passer à l'élément suivant, vous incrémentez l'index variable en y ajoutant la valeur 1, puis sélectionnez-la dans la liste en utilisant cette valeur incrémentée. La figure 19-5 montre les blocs permettant de procéder à cette opération.

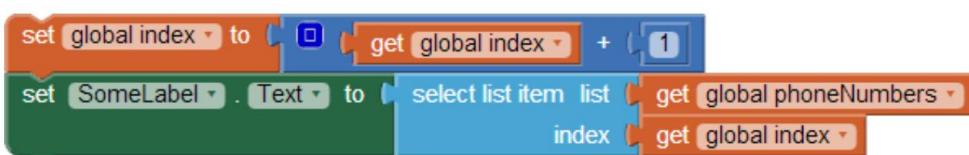


Figure 19-6. Incrémentation de la valeur d'index et utilisation de la valeur incrémentée pour sélectionner l'élément de liste suivant

Exemple : Parcourir une liste de couleurs de peinture

Prenons un exemple d'application avec laquelle l'utilisateur peut parcourir chaque couleur de peinture potentielle pour sa maison en appuyant sur un « ColorButton ». Chaque fois que l'utilisateur appuie, la couleur du bouton change. Lorsque l'utilisateur parcourt toutes les couleurs possibles, l'application revient à la première.

Pour cet exemple, nous utiliserons quelques couleurs de base. Cependant, vous pouvez personnaliser les blocs de code pour parcourir n'importe quel ensemble de couleurs.

Votre première étape consiste à définir une variable de liste pour la liste de couleurs et à l'initialiser avec des couleurs de peinture comme éléments, comme illustré dans la figure 19-6.



Figure 19-7. Initialisation de la liste des couleurs avec une liste de couleurs de peinture

Ensuite, définissez une variable d'index qui suit la position actuelle dans la liste. Il doit commencer à 1. Vous pouvez donner à la variable un nom descriptif tel que `currentColorIndex`, mais si vous ne gérez pas plusieurs index dans votre application, vous pouvez simplement la nommer « `index` », comme dans la figure 19-4.

L'utilisateur passe à la couleur suivante dans la liste en cliquant sur le `ColorButton`. À chaque pression, l'index doit être incrémenté et la couleur d'arrière-plan du bouton doit changer pour l'élément actuellement sélectionné, comme le montre la figure 19-7.



Figure 19-8. Chaque pression sur le bouton change de couleur

Supposons que l'arrière-plan du bouton soit initialement défini sur Rouge dans le Concepteur de composants. La première fois que l'utilisateur appuie sur le bouton, l'index passe de sa valeur initiale de 1 à 2 et la couleur d'arrière-plan du bouton passe au deuxième élément de la liste, le vert. La deuxième fois que l'utilisateur appuie dessus, l'index passe de 2 à 3 et la couleur d'arrière-plan passe au bleu.

Mais à votre avis, que se passera-t-il la prochaine fois que l'utilisateur appuiera dessus ?

Si vous aviez dit qu'il y aurait une erreur, vous avez raison ! l'index deviendra 4 et l'application tentera de sélectionner le quatrième élément de la liste, mais la liste ne contient que trois éléments. L'application forcera la fermeture ou quittera, et l'utilisateur verra un message d'erreur comme celui de la figure 19-8.

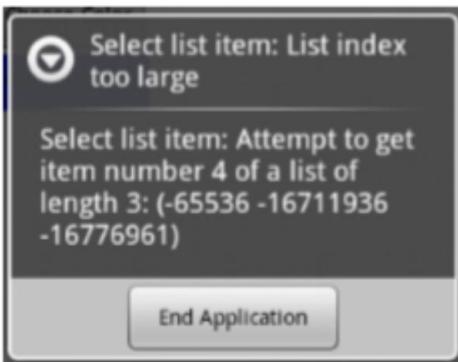


Figure 19-9. Le message d'erreur affiché lorsque l'application tente de sélectionner un quatrième élément dans une liste de trois éléments

De toute évidence, ce message n'est pas quelque chose que vous souhaitez que les utilisateurs de votre application voient. Pour éviter ce problème, ajoutez un bloc if pour vérifier si la dernière couleur de la liste a été atteinte. Si c'est le cas, l' index peut être ramené à 1 afin que la première couleur soit à nouveau affichée, comme illustré dans la figure 19-9.

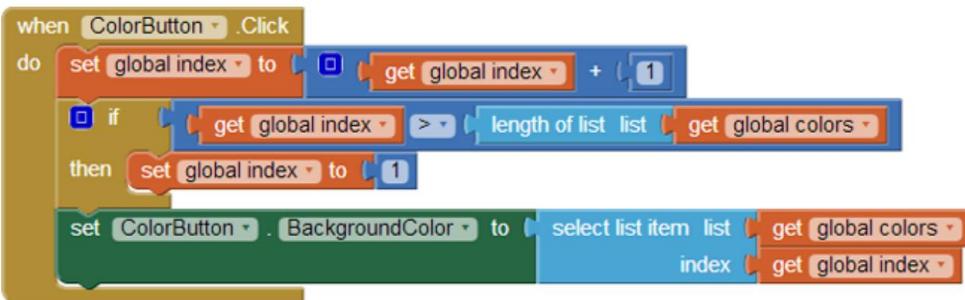


Figure 19-10. Utiliser un if pour vérifier si la valeur de l'index est supérieure à la longueur de la liste

Lorsque l'utilisateur appuie sur le bouton, l' index est incrémenté puis vérifié pour voir si sa valeur est trop grande. L' indice est comparé à la longueur de la liste, et non à 3 ; de cette façon, votre application fonctionnera même si vous ajoutez des éléments à la liste. En vérifiant si l'index est supérieur à la longueur de votre liste (au lieu de vérifier s'il est supérieur au nombre spécifique 3), vous avez éliminé une dépendance de code dans votre application. Une dépendance de code est un terme de programmation qui décrit un code défini de manière trop spécifique et manquant de flexibilité. Ainsi, si vous modifiez quelque chose à un seul endroit (dans notre exemple ici, vous ajoutez des éléments à votre liste), vous devrez rechercher chaque instance où vous utilisez cette liste et la modifier explicitement.

Comme vous pouvez l'imaginer, ce type de dépendances peut devenir compliqué très rapidement, et ils conduisent généralement à beaucoup plus de bugs que vous devez également rechercher. En fait, le

La conception de notre application Color contient une autre dépendance de code telle qu'elle est actuellement programmée. Pouvez-vous comprendre ce que c'est ?

Si vous avez changé la première couleur de votre liste du rouge à une autre couleur, l'application ne fonctionnera pas correctement à moins que vous n'ayez également pensé à modifier le Button.BackgroundColor initial que vous avez défini dans le Concepteur de composants. La façon d'éliminer cette dépendance de code consiste à définir le ColorButton.BackgroundColor initial sur la première couleur de la liste plutôt que sur une couleur spécifique. Étant donné que ce changement implique un comportement qui se produit lors de la première ouverture de votre application, vous effectuez cette opération dans le gestionnaire d'événements Screen.Initialize qui est invoqué lors du lancement d'une application, comme illustré dans la figure 19-10.



Figure 19-11. Définir la BackgroundColor du bouton sur la première couleur de la liste au lancement de l'application

Création de formulaires de saisie et de données dynamiques

L'application Color précédente impliquait une liste statique : une liste dont les éléments sont définis par le programmeur (vous) et dont les éléments ne changent pas sauf si vous modifiez les blocs eux-mêmes. Le plus souvent, cependant, les applications traitent des données dynamiques : des informations qui changent en fonction de la saisie de nouveaux éléments par l'utilisateur final ou du chargement de nouveaux éléments à partir d'une base de données ou d'une source d'informations Web. Dans cette section, nous discutons d'un exemple d'application Note Taker, dans laquelle l'utilisateur saisit des notes dans un formulaire et peut afficher toutes ses notes précédentes.

Définir une liste dynamique

Des applications telles que Note Taker commencent par une liste vide. Lorsque vous voulez une liste qui commence vide, vous la définissez avec le bloc créer une liste vide , comme illustré dans la figure 19-11.



Figure 19-12. Les blocs pour définir une liste dynamique ne contiennent aucun élément prédéfini

Ajouter un article

La première fois que quelqu'un lance l'application, la liste des notes est vide. Mais lorsque l'utilisateur saisit des données dans un formulaire et appuie sur Soumettre, de nouvelles notes seront ajoutées à la liste. Le formulaire peut être aussi simple que celui illustré à la figure 19-12.



Figure 19-13. Utiliser un formulaire pour ajouter de nouveaux éléments à la liste de notes

Lorsque l'utilisateur saisit une note et appuie sur le bouton Soumettre, l'application appelle l'ajout La fonction items to list permet d'ajouter le nouvel élément à la liste, comme illustré dans la figure 19-13.



Figure 19-14. Appel d'ajouter des éléments à la liste pour ajouter la nouvelle note lorsque l'utilisateur appuie sur le bouton Submit

Vous pouvez trouver le bloc Ajouter des éléments à la liste dans le tiroir Liste. Attention : il y a également un bloc d'ajout à la liste , mais celui-ci est un bloc assez rare utilisé pour ajouter une liste entière à une autre.

Afficher une liste

Le contenu des variables de liste, comme toutes les variables, n'est pas visible par l'utilisateur. Les blocs de la figure 19-13 ajoutent des éléments à la liste à chaque fois que SubmitButton.Click est invoqué, mais l'utilisateur ne recevra pas d'information indiquant que la liste s'agrandit tant que vous n'aurez pas programmé davantage de blocs pour afficher réellement le contenu de la liste.

La manière la plus simple d'afficher une liste dans l'interface utilisateur de votre application consiste à utiliser la même méthode que vous utilisez pour afficher des nombres et du texte : placez la liste dans la propriété Text d'un composant Label , comme illustré dans la figure 19-14.



Figure 19-15. Afficher la liste à l'utilisateur en la plaçant dans une étiquette.

Malheureusement, cette méthode simple d'affichage d'une liste n'est pas très élégante ; ça met le liste entre parenthèses, chaque élément étant séparé par un espace et pas nécessairement sur la même ligne. Par exemple, si l'utilisateur tape : « Est-ce que je terminerai un jour ce livre ? comme première note, et "J'oublie à quoi ressemble mon fils!" dans le second cas, l'application afficherait la liste de notes similaire à celle que nous voyons dans la figure 19-15.



Figure 19-16. Ces entrées sont répertoriées en utilisant le formatage par défaut

Au chapitre 20, vous pouvez découvrir une manière plus sophistiquée d'afficher une liste.

Supprimer un élément d'une liste

Vous pouvez supprimer un élément d'une liste à l'aide du bloc Supprimer un élément de liste , comme illustré dans la figure 19-16.



Figure 19-17. Supprimer un élément d'une liste

Les blocs de la figure 19-16 suppriment le deuxième élément de la liste nommée notes.

En règle générale, cependant, vous ne souhaiterez pas supprimer un élément modifié (par exemple, 2), mais fournirez plutôt un mécanisme permettant à l'utilisateur de choisir l'élément à supprimer.

Vous pouvez utiliser le composant ListPicker pour fournir à l'utilisateur un moyen de sélectionner un article. ListPicker est livré avec un bouton associé. Lorsque vous appuyez sur le bouton, ListPicker affiche les éléments d'une liste dans laquelle l'utilisateur peut en choisir un. Lorsque l'utilisateur choisit un élément, l'application peut le supprimer.

ListPicker est facile à programmer si vous comprenez ses événements clés, BeforePicking et AfterPicking et ses propriétés clés, Elements, Selection et SelectionIndex (voir Tableau 19-1).

Tableau 19-1. Les événements et propriétés clés du composant ListPicker

Événement	Propriété
BeforePicking : déclenché lorsque le bouton est cliqué. Éléments : la liste de choix.	
AfterPicking : déclenché lorsque l'utilisateur fait un choix. Sélection : Le choix de l'utilisateur. SelectionIndex : Position de choix.	

L'utilisateur déclenche l' événement ListPicker.BeforePicking en appuyant sur le bouton ListPicker. bouton associé. Dans le gestionnaire d'événements ListPicker.BeforePicking , vous allez définir la propriété ListPicker.Elements sur une variable de liste afin que les données de la liste s'affichent. Pour l'application Note Taker, vous devez définir Elements sur la variable notes qui contient votre liste de notes, comme le montre la figure 19-17.

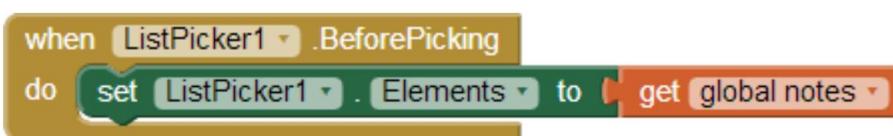


Figure 19-18. La propriété Elements de ListPicker1 est définie sur la liste de notes

Avec ces blocs, les éléments de la liste des notes apparaîtront dans le ListPicker. S'il y avait deux notes, elles apparaîtraient comme le montre la figure 19-18.

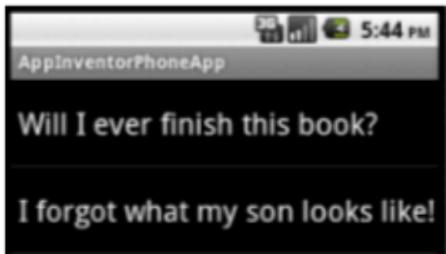


Figure 19-19. La liste des notes apparaît dans le ListPicker

Lorsque l'utilisateur choisit un élément dans la liste, il déclenche le ListPicker.AfterPicking événement. Dans ce gestionnaire d'événements, vous pouvez accéder à la sélection de l'utilisateur dans la propriété ListPicker.Selection .

Cependant, votre objectif dans cet exemple est de supprimer un élément de la liste, et le bloc Supprimer l'élément de la liste attend un index, pas un élément. La propriété Selection de ListPicker correspond aux données réelles (la note), pas à l'index. Par conséquent, vous devez plutôt utiliser la propriété SelectionIndex car elle vous fournit l'index de l'élément choisi. Il doit être défini comme index du bloc de suppression d'élément de liste , comme le montre la figure 19-19.



Figure 19-20. Suppression d'un élément à l'aide de ListPicker.SelectionIndex

Listes de listes

Les éléments d'une liste peuvent être de n'importe quel type, notamment des nombres, du texte, des couleurs ou des valeurs booléennes (vrai/faux). Mais les éléments d'une liste peuvent aussi être eux-mêmes des listes. Des structures de données aussi complexes sont courantes. Par exemple, une liste de listes pourrait être utilisée pour convertir le Quiz des Présidents (Chapitre 8) en un quiz à choix multiples. Revenons à l'essentiel

structure du Quiz des présidents, qui est une liste de questions et une liste de réponses, comme le montre la figure 19-20.

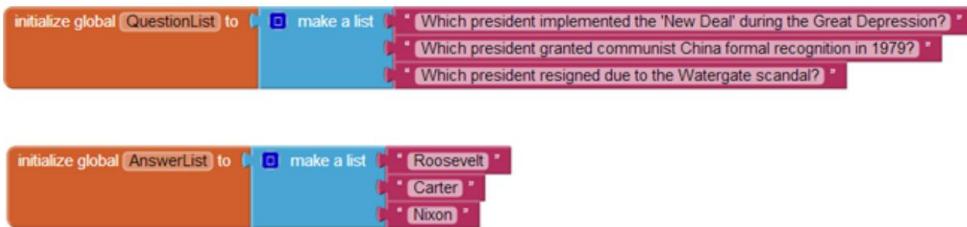


Figure 19-21. Une liste de questions et une liste de réponses

Chaque fois que l'utilisateur répond à une question, l'application vérifie si elle est correcte en comparant la réponse à l'élément actuel dans AnswerList.

Pour que le quiz soit à choix multiples, vous devez conserver une liste supplémentaire, dans laquelle sont stockés les choix pour chaque réponse à chaque question. Vous spécifiez ces données en plaçant trois blocs de création de liste dans un bloc de création de liste interne , comme le montre la figure 19-21.



Figure 19-22. Une liste de listes est formée en insérant des blocs de création de liste en tant qu'éléments dans un bloc de liste interne.

Chaque élément de la variable AnswerChoices est lui-même une liste contenant trois éléments. Si vous sélectionnez un élément dans AnswerChoices, le résultat est une liste. Maintenant que vous avez renseigné vos réponses à choix multiples, comment les afficheriez-vous à l'utilisateur ?

Comme avec l'application Note Taker, vous pouvez utiliser un ListPicker pour présenter les choix à l'utilisateur. Si l'index était nommé currentQuestionIndex, l' événement

ListPicker.BeforePicking apparaîtrait comme indiqué dans la figure 19-22.

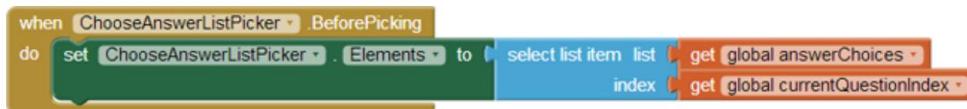


Figure 19-23. Utilisation du sélecteur de liste pour présenter l'une des listes de choix de réponses au utilisateur

Ces blocs prendraient la sous-liste actuelle de choix de réponses et permettraient à l'utilisateur de choisir. Ainsi, si currentQuestionIndex valait 1, ListPicker afficherait une liste comme celle de la figure 19-23.

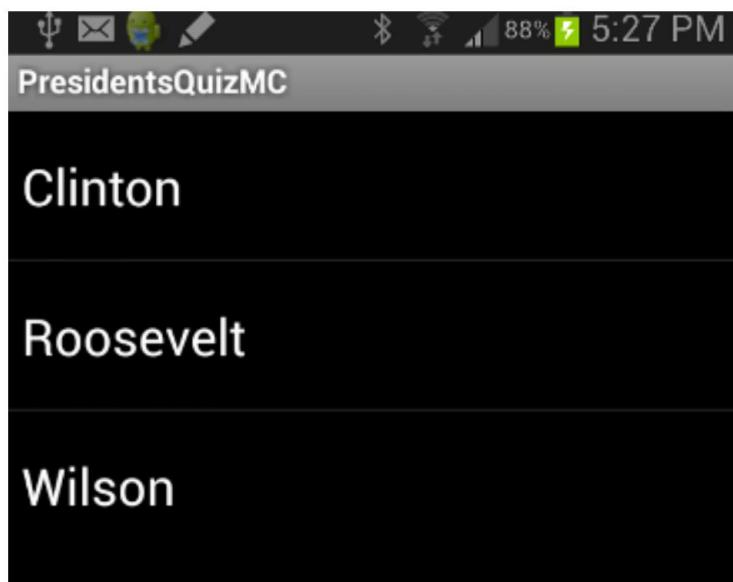


Figure 19-24. Les choix de réponses présentés à l'utilisateur pour la première question

Lorsque l'utilisateur choisit, vous vérifiez la réponse avec les blocs illustrés dans la figure 19-24.

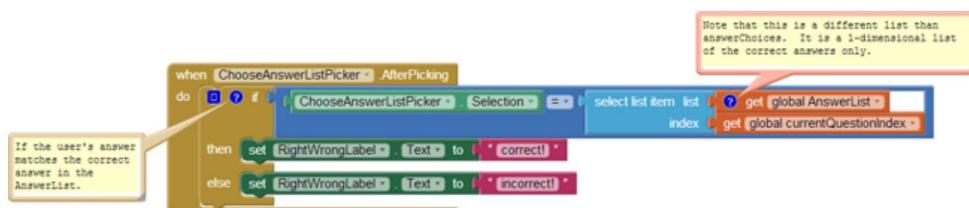


Figure 19-25. Vérifier si l'utilisateur a choisi la bonne réponse

Dans ces blocs, la sélection de l'utilisateur dans ListPicker est comparée à la bonne réponse, qui est stockée dans une liste différente, AnswerList (car AnswerChoices ne fournit que les choix et n'indique pas la bonne réponse).

Résumé

Les listes sont utilisées dans presque toutes les applications auxquelles vous pouvez penser. Comprendre leur fonctionnement est fondamental pour la programmation. Dans ce chapitre, nous avons exploré l'un des modèles de programmation les plus courants : l'utilisation d'une variable d'index qui commence au début de la liste et est incrémentée jusqu'à ce que chaque élément de la liste soit traité. Si vous pouvez comprendre et personnaliser ce modèle, vous êtes effectivement un programmeur !

Nous avons ensuite abordé certains des autres mécanismes de manipulation de liste, y compris les formulaires typiques permettant à l'utilisateur d'ajouter et de supprimer des éléments. Une telle programmation nécessite encore un autre niveau d'abstraction, car vous devez visualiser les données dynamiques avant qu'elles n'existent réellement. Après tout, vos listes sont vides jusqu'à ce que l'utilisateur y mette quelque chose. Si vous comprenez cela, vous pourriez même envisager de quitter votre emploi quotidien.

Nous avons conclu le chapitre en introduisant une structure de données complexe, une liste de listes. Il s'agit certainement d'un concept difficile, mais nous l'avons exploré en utilisant des données fixes : les choix de réponses pour un quiz à choix multiples. Si vous maîtrisez cela ainsi que le reste du chapitre, votre test final est le suivant : créez une application qui utilise une liste de listes mais avec des données dynamiques. Un exemple serait une application avec laquelle les gens peuvent créer leurs propres quiz à choix multiples, étendant encore plus l'application MakeQuiz du chapitre 10. Bonne chance !

Pendant que vous réfléchissez à la manière dont vous allez aborder ce problème, comprenez que notre exploration de les listes ne sont pas terminées. Dans le chapitre suivant, nous poursuivons la discussion et nous concentrerons sur l'itération de liste avec une touche d'originalité : appliquer des fonctions à chaque élément d'une liste.

Blocs répétitifs

S'il y a une chose pour laquelle les ordinateurs sont doués, c'est bien la répétition des choses : comme les petits enfants, ils ne se lassent jamais de répéter. Ils sont également très rapides et peuvent notamment traiter l'intégralité de votre liste d'amis Facebook en une microseconde.

Dans ce chapitre, vous apprendrez à programmer la répétition avec des blocs de répétition spéciaux au lieu de copier et coller les mêmes blocs encore et encore. Vous apprendrez comment envoyer un SMS à chaque numéro de téléphone d'une liste et comment ajouter une liste de numéros. Vous apprendrez également que les blocs répétés peuvent simplifier considérablement une application.

Figure 20-1.



Contrôler l'exécution d'une application : branchement et Boucle

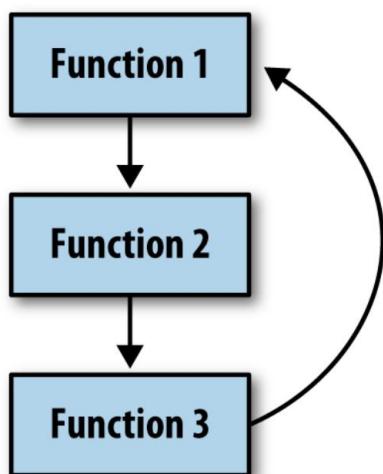


Figure 20-2. Les blocs répétés provoquent la boucle d'un programme

Dans les chapitres précédents, vous avez appris que vous définir le comportement d'une application avec un ensemble de gestionnaires d'événements : les événements et les fonctions qui devraient être exécuté en réponse. Vous avez également appris que la réponse à un événement n'est souvent pas une séquence linéaire de fonctions et peut contenir des blocs qui ne sont exécutés que sous certaines conditions.

Les blocs de répétition sont l'autre manière par laquelle un application se comporte de manière non linéaire. Tout comme si et sinon si les blocs permettaient à un programme de créer une branche, les blocs répétitifs permettent à un programme de boucler ; c'est-à-dire exécuter un ensemble de fonctions, puis remonter dans le code et recommencer, comme illustré dans la figure 20-1. Lorsqu'une application s'exécute, un compteur de programme fonctionnant sous le capot de l'application assure le suivi de la prochaine opération à effectuer. Jusqu'à présent, vous avez examiné des applications dans lesquelles le compteur de programme démarre en haut d'un gestionnaire d'événements et (sous réserve)

effectue des opérations de haut en bas. Avec les blocs répétés, le compteur de programme effectue une boucle dans les blocs, effectuant continuellement les mêmes opérations.

App Inventor fournit un certain nombre de blocs de répétition, notamment `for each` et `while`, sur lesquels nous nous concentrerons dans ce chapitre. `foreach` est utilisé pour spécifier les fonctions qui doivent être exécutées sur chaque élément d'une liste. Ainsi, si vous avez une liste de numéros de téléphone, vous pouvez spécifier qu'un SMS soit envoyé à chaque numéro de la liste.

Le bloc `while` est plus général que le bloc `for each`. Avec lui, vous pouvez programmer des blocs qui se répètent continuellement jusqu'à ce qu'une condition arbitraire change. Vous pouvez utiliser les blocs `while` pour calculer des formules mathématiques telles que l'addition des n premiers nombres ou le calcul de la factorielle de n. Vous pouvez également utiliser `while` lorsque vous devez traiter deux listes simultanément ; pour chacun , il ne traite qu'une seule liste à la fois.

Itérer des fonctions sur une liste avec pour chacune

Le chapitre 18 présente une application qui appelle de manière aléatoire un numéro de téléphone dans une liste. Appeler un ami au hasard peut parfois fonctionner, mais si vous avez des amis comme le mien, ils ne répondent pas toujours. Une stratégie différente serait d'envoyer un message « Je pense à vous ! » envoyez un SMS à tous vos amis et voyez qui répond en premier (ou de la manière la plus charmante !).

Une façon de mettre en œuvre une telle application consiste simplement à copier les blocs pour envoyer des SMS à un seul utilisateur. numéro, puis collez-les pour chaque ami à qui vous souhaitez envoyer un SMS, comme indiqué dans la figure 20-2.

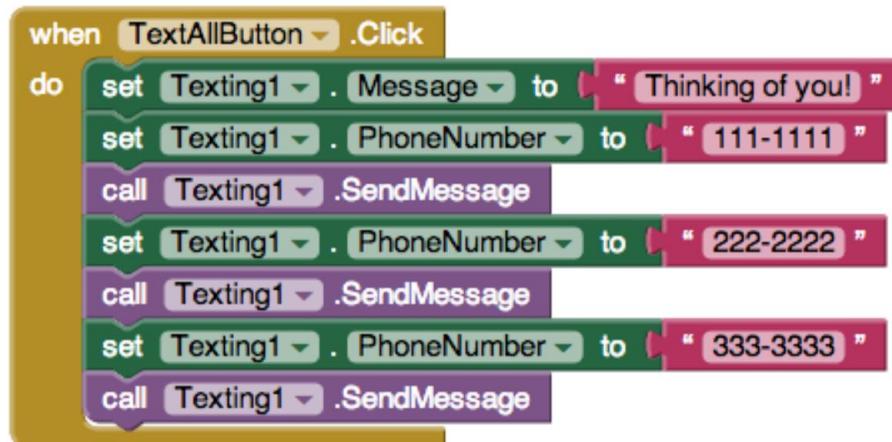


Figure 20-3. Copier et coller les blocs pour chaque numéro de téléphone à envoyer par SMS

Cette méthode de copier-coller « force brute » convient parfaitement si vous n'avez que quelques blocs à répéter. Mais si vous traitez de grandes quantités de données ou des données qui vont changer, vous ne le ferez pas.

souhaitez modifier votre application avec la méthode copier-coller chaque fois que vous ajoutez ou supprimez un numéro de téléphone de votre liste.

Le for each block offre une meilleure solution. Vous définissez une variable phoneNumbers avec tous les nombres, puis enroulez un pour chaque bloc autour d'une seule copie des blocs que vous souhaitez exécuter. La figure 20-3 montre pour chaque solution d'envoi de SMS à un groupe.

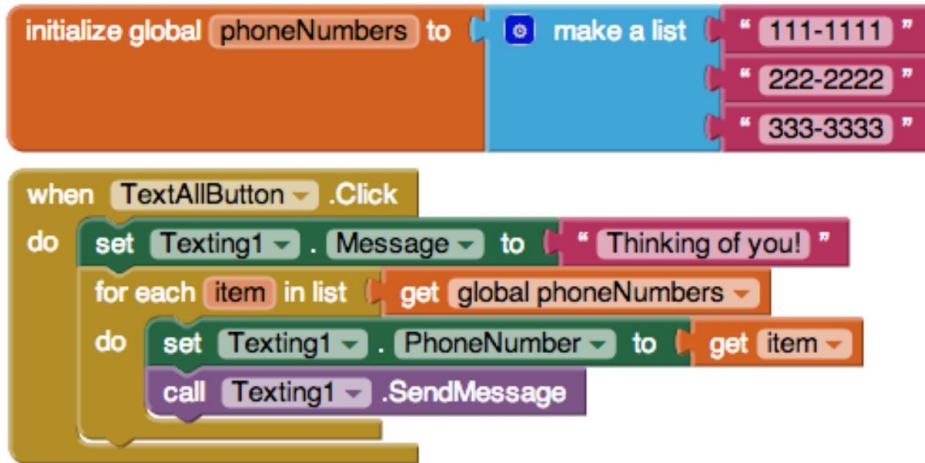


Figure 20-4. Utiliser le for each block pour effectuer les mêmes blocs pour chaque élément de la liste

Vous pouvez lire ce code comme suit : « Pour chaque élément (numéro de téléphone) de la liste des numéros de téléphone, définissez le numéro de téléphone de l'objet Texting sur l'élément et envoyez le message texte.

En haut de chaque bloc, vous précisez la liste qui sera traitée. Le bloc dispose également d'une variable d'espace réservé fournie avec le pour chacun. Par défaut, cet espace réservé est nommé « élément ». Vous pouvez le laisser ainsi ou le renommer. Cette variable représente l'élément en cours de traitement dans la liste.

Si une liste comporte trois éléments, les blocs internes seront exécutés trois fois. On dit que les blocs internes sont subordonnés ou imbriqués dans chaque bloc. Nous disons que le compteur de programme « boucle » lorsqu'il atteint le bloc inférieur dans le pour chaque.

Un examen plus approfondi du bouclage

Examinons en détail les mécanismes des blocs for each , car la compréhension des boucles est fondamentale en programmation. Lorsque l'utilisateur appuie sur TextAllButton et sur le

gestionnaire d'événements est invoqué, la première opération exécutée est la définition de Texting1.Message à bloquer, qui définit le message sur « Je pense à vous ! » Ce bloc n'est exécuté qu'une seule fois.

Le pour chaque bloc commence alors. Avant que les blocs imbriqués de a for each ne soient exécutés, l'élément variable d'espace réservé est défini sur le premier numéro de la liste phoneNumbers (111-1111). Cela se produit automatiquement ; le for each vous évite d'avoir à appeler manuellement un élément de liste de sélection. Une fois le premier élément sélectionné dans l'élément variable, les blocs contenus dans chacun sont exécutés pour la première fois. Le

La propriété Texting1.PhoneNumber est définie sur la valeur de l'élément (111-1111) et le message est envoyé.

Après avoir atteint le dernier bloc dans a for each (le bloc Texting.SendMessage), l'application « boucle » vers le haut de for each et place automatiquement l'élément suivant de la liste (222-2222) dans l' élément variable. Les deux opérations dans le pour chacune sont ensuite répétées, envoyant le message « Je pense à vous ! » envoyez un SMS au 222-2222. L'application effectue ensuite une nouvelle boucle et définit l'élément sur le dernier élément de la liste (333 à 3333). Les opérations sont répétées une troisième fois, envoyant le troisième texte.

Étant donné que le dernier élément de la liste a été traité, la boucle for each s'arrête à ce point. Dans le jargon de programmation, nous disons que le contrôle "sort" de la boucle, ce qui signifie que le compteur du programme continue de traiter les blocs situés en dessous de chacun. Dans cet exemple, il n'y a aucun bloc en dessous, donc le gestionnaire d'événements se termine.

Écrire du code maintenable

Pour l'utilisateur de l'application, la solution pour chaque solution qui vient d'être décrite se comporte exactement de la même manière que la méthode « force brute » consistant à copier puis coller les blocs de texte. Du point de vue du programmeur, cependant, la solution for each est plus maintenable et peut être utilisée même si les données (la liste téléphonique) sont saisies dynamiquement.

Un logiciel maintenable est un logiciel qui peut être modifié facilement sans introduire de bogues. Avec la solution for each , vous pouvez modifier la liste des amis qui reçoivent des SMS en modifiant uniquement la variable list. Vous n'avez pas du tout besoin de modifier la logique de votre programme (le gestionnaire d'événements). Comparez cela avec la méthode par force brute, qui vous oblige à ajouter de nouveaux blocs dans le gestionnaire d'événements lorsqu'un nouvel ami est ajouté.

Chaque fois que vous modifiez la logique d'un programme, vous risquez d'introduire des bugs.

Tout aussi important, la solution for each fonctionnerait même si la liste téléphonique était dynamique ; c'est-à-dire celui dans lequel l'utilisateur final peut ajouter des numéros à la liste. Contrairement à notre exemple, qui comporte trois numéros de téléphone particuliers répertoriés dans le code, la plupart des applications fonctionnent avec des données dynamiques provenant de l'utilisateur final ou d'une autre source. Si vous avez repensé cette application pour que l'utilisateur final puisse saisir les numéros de téléphone, vous devrez utiliser un pour chaque solution, car lorsque vous écrivez le programme, vous ne savez pas quels numéros mettre dans la solution par force brute.

Utiliser pour chacun pour afficher une liste

Lorsque vous souhaitez afficher les éléments d'une liste sur le téléphone, vous pouvez brancher la liste dans la propriété Text d'une étiquette, comme le montre la figure 20-4.

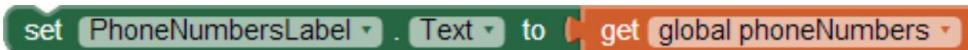


Figure 20-5. La manière simple d'afficher une liste est de la brancher directement sur une étiquette

Lorsque vous branchez une liste directement dans une propriété Text d'une étiquette, les éléments de la liste sont affichés dans l'étiquette sous la forme d'une seule ligne de texte, séparés par des espaces et contenus entre parenthèses :

(111-1111 222-2222 333-3333)

Les nombres peuvent ou non s'étendre sur plus d'une ligne, selon le nombre. L'utilisateur peut voir les données et peut-être comprendre qu'il s'agit d'une liste de numéros de téléphone, mais ce n'est pas très élégant. Les éléments de liste sont généralement affichés sur des lignes séparées ou avec des virgules les séparant.

Pour afficher correctement une liste, vous avez besoin de blocs qui transforment chaque élément de la liste en un seul valeur de texte avec le formatage souhaité. Les objets texte sont généralement constitués de lettres, de chiffres et de signes de ponctuation. Cependant, le texte peut également stocker des caractères de contrôle spéciaux, qui ne correspondent pas à un caractère que vous pouvez voir. Un onglet, par exemple, est noté \t. (Pour en savoir plus sur les caractères de contrôle, consultez la norme Unicode pour la représentation de texte sur <http://www.unicode.org/standard/standard.html>.)

Dans notre liste de numéros de téléphone, nous voulons un caractère de nouvelle ligne, désigné par \n.

Lorsque \n apparaît dans un bloc de texte, cela signifie « passer à la ligne suivante avant d'afficher l'élément suivant ». Ainsi, l'objet texte « 111-1111\n222-2222\n333-3333 » apparaîtrait comme :

111-1111

222-2222

333-3333

Pour créer un tel objet texte, nous utilisons un pour chaque bloc et « traitons » chaque élément en l'ajoutant avec un caractère de nouvelle ligne à la propriété PhoneNumberLabel.Text , comme le montre la figure 20-5.

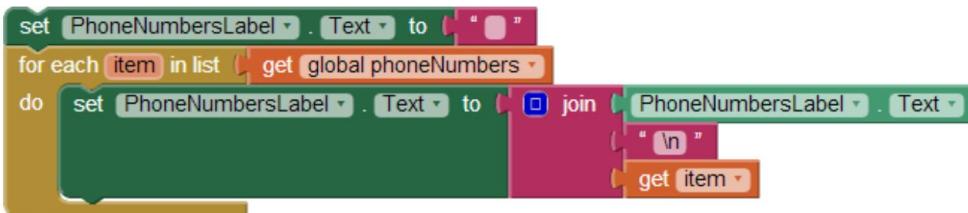


Figure 20-6. Un pour chaque bloc utilisé pour afficher une liste avec des éléments sur des lignes séparées

314 Chapitre 20 : Répétition de blocs

Traçons les blocs pour voir comment ils fonctionnent. Comme indiqué au chapitre 15, le traçage montre comment chaque variable ou propriété change à mesure que les blocs sont exécutés. Avec un pour chacun, nous considérons les valeurs après chaque itération ; c'est-à-dire que chaque fois que le programme parcourt la boucle for each .

Avant le pour chacun, le PhoneNumbersLabel est initialisé avec le texte vide. Lorsque le for each commence, l'application place automatiquement le premier élément de la liste (111-1111) dans l' élément variable d'espace réservé. Les blocs dans for each se joignent ensuite à PhoneNumbersLabel.Text (le texte vide), \n et l'élément, et définissent le résultat dans PhoneNumbersLabel.Text. Ainsi, après la première itération du for each, les variables pertinentes stockent les valeurs indiquées dans le tableau 20-1.

Tableau 20-1. Les valeurs après la première itération

article	PhoneNumbersLabel.Text
111-1111	\n111-1111

Parce que le bas du for each a été atteint, les boucles de contrôle remontent et l'élément suivant de la liste (222-2222) est placé dans l' élément variable. Lorsque les blocs internes sont répétés, le texte concatène la valeur de PhoneNumbersLabel.Text (\n111-1111) avec \n, puis avec l'élément, qui est maintenant 222-2222. Après cette deuxième itération, les variables stockent les valeurs indiquées dans le tableau 20-2.

Tableau 20-2. Les valeurs après la deuxième itération

article	PhoneNumbersLabel.Text
222-2222	\n111-1111\n222-2222

Le troisième élément de la liste est ensuite placé dans item, et le bloc interne est répété un troisième fois. La valeur finale des variables, après cette dernière itération, est affichée dans Tableau 20-3.

Tableau 20-3. Les valeurs des variables après la dernière itération

article	PhoneNumbersLabel.Text
333-3333	\n111-1111\n222-2222\n333-3333

Ainsi, après chaque itération, l'étiquette devient plus grande et contient un numéro de téléphone supplémentaire (et une nouvelle ligne supplémentaire). À la fin de chaque, PhoneNumbersLabel.Text est défini de manière à ce que les numéros apparaissent comme suit :

111-1111.

222-2222

333-3333

Le bloc while-do

Le bloc while-do est un peu plus compliqué à utiliser que pour chacun. L'avantage du bloc while-do réside dans sa généralité : car each se répète sur une liste, mais while peut se répéter tant que n'importe quelle condition arbitraire est vraie.

Comme vous l'avez appris au chapitre 18, une condition teste quelque chose et renvoie une valeur vraie ou fausse. Les blocs while-do incluent un test conditionnel, tout comme les blocs if . Si le test d'un certain temps est évalué à vrai, l'application exécute les blocs internes, puis effectue une boucle et revérifie le test. Tant que le test est vrai, les blocs internes sont répétés. Lorsque le test est évalué comme faux, l'application sort de la boucle (comme nous l'avons vu avec le bloc for each) et continue avec les blocs situés sous le while-do.

Utiliser while-do pour calculer une formule

Voici un exemple de bloc while-do qui répète des opérations. À votre avis, que font les blocs de la figure 20-6 ? Une façon de comprendre cela est de tracer chaque bloc (voir le chapitre 15 pour en savoir plus sur le traçage), en suivant la valeur de chaque variable au fur et à mesure.

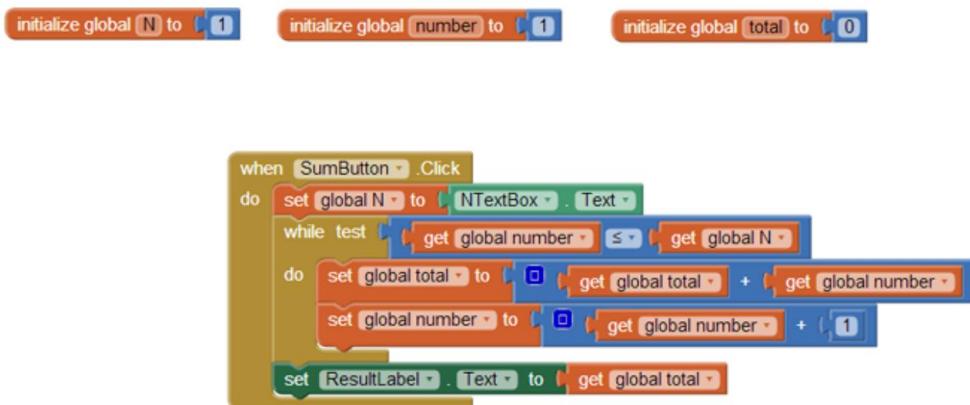


Figure 20-7. Pouvez-vous comprendre ce que font ces blocs ?

316 Chapitre 20 : Répétition de blocs

Les blocs de la boucle while-do seront répétés tant que le numéro de variable est inférieur ou égal à la variable N. Pour cette application, N est défini sur un nombre que l'utilisateur final saisit dans une zone de texte (NTextBox). Supposons que l'utilisateur tape un 3. Les variables de l'application ressembleraient au tableau 20-4 lorsque le bloc while-do est atteint pour la première fois.

Tableau 20-4. Valeurs variables lorsque while-do est atteint pour la première fois

N	nombre total
3	0

Le bloc while-do teste d'abord la condition : le nombre est -il inférieur ou égal à (\leq) N ? La première fois que cette question est posée, le test est vrai, donc les blocs imbriqués dans le bloc while-do sont exécutés. le total est défini sur lui-même (0) plus le nombre (1), et le nombre est incrémenté. Après la première itération des blocs dans le while-do, la variable les valeurs sont celles indiquées dans le tableau 20-5.

Tableau 20-5. Les valeurs des variables après la première itération des blocs dans le bloc while

N	nombre total
3	1

A la deuxième itération, le test « nombre \leq N » est toujours vrai ($2 \leq 3$), donc les blocs internes sont à nouveau exécutés. le total est défini sur lui-même (1) plus le nombre (2). le numéro est incrémenté. Une fois cette deuxième itération terminée, les variables conservent les valeurs répertoriées dans le tableau 20-6.

Tableau 20-6. Les valeurs des variables après la deuxième itération

N	nombre total
3	3

L'application effectue une nouvelle boucle et teste la condition. Encore une fois, c'est vrai ($3 \leq 3$), donc les blocs sont exécutés une troisième fois. Maintenant, le total est défini sur lui-même (3) plus le nombre (3), il devient donc 6. Le nombre est incrémenté jusqu'à 4, comme indiqué dans le tableau 20-7.

Tableau 20-7. Les valeurs après la troisième itération

N	nombre total
3 4	6

Après cette troisième itération, l'application revient une fois de plus au début du temps-faire. Lorsque le test « `nombre≤N` » s'exécute cette fois, il teste $4 \leq 3$, ce qui est évalué comme faux. Ainsi, les blocs imbriqués du `while-do` ne sont pas exécutés à nouveau et le gestionnaire d'événements se termine.

Alors, à quoi servent ces blocs ? Ils ont effectué l'une des opérations mathématiques les plus fondamentales : compter les nombres. Quel que soit le nombre saisi par l'utilisateur, l'application affichera la somme des nombres `1..N`, où N est le nombre saisi. Dans cet exemple, N vaut 3, donc l'application a obtenu un total de $1+2+3=6$. Si l'utilisateur avait tapé 4, l'application aurait calculé 10.

Résumé

Les ordinateurs savent très bien répéter la même fonction encore et encore. Pensez à tous les comptes bancaires qui sont traités pour générer des intérêts, à toutes les notes traitées pour calculer la moyenne des notes des élèves et aux innombrables autres exemples quotidiens pour lesquels les ordinateurs utilisent la répétition pour effectuer une tâche.

Ce chapitre a exploré deux des blocs de répétition d'App Inventor. Le `pour chaque bloc` applique un ensemble de fonctions à chaque élément d'une liste. En l'utilisant, vous pouvez concevoir un code de traitement qui fonctionne sur une liste abstraite plutôt que sur des données concrètes. Un tel code est plus maintenable ; et si les données à traiter sont dynamiques, c'est obligatoire.

Par rapport à chacun, `while-do` est plus général : vous pouvez l'utiliser pour traiter une liste, mais vous pouvez également l'utiliser pour traiter de manière synchrone deux listes ou calculer une formule. Avec `while-do`, les blocs internes sont exécutés en continu aussi longtemps qu'une certaine condition est vraie. Une fois les blocs exécutés pendant ce temps, le contrôle est sauvegardé et la condition de test est réessayée. Ce n'est que lorsque le test est évalué comme faux que le bloc `while-do` se termine.

Définition des procédures et réutilisation des blocs

Figure 21-1.



Les langages de programmation tels qu'App Inventor fournissent un ensemble de base de fonctionnalités intégrées : dans le cas d'App Inventor, un ensemble de blocs de base. Les langages de programmation offrent également un moyen d'étendre cette fonctionnalité en ajoutant de nouvelles fonctions (blocs) au langage. Dans App Inventor, vous faites cela en définissant des procédures (appelées séquences de blocs) que votre application peut appeler de la même manière qu'elle appelle les blocs prédéfinis d'App Inventor. Comme vous le verrez dans ce chapitre, être capable de créer de telles abstractions est très important pour résoudre des problèmes complexes, ce qui constitue la pierre angulaire de la création d'applications vraiment convaincantes.

Lorsque les parents disent à leur enfant : « Va te brosser les dents avant de te coucher », ils veulent en réalité dire : « Sortez votre brosse à dents et votre dentifrice du placard, versez un peu de dentifrice sur la brosse, faites pivoter la brosse sur

chaque dent pendant 10 secondes (ha !), et ainsi de suite. « Brossez-vous les dents » est une abstraction : un nom reconnaissable pour une séquence d'instructions de niveau inférieur. Dans ce cas, les parents demandent à l'enfant d'exécuter les instructions qui, selon eux, signifient « se brosser les dents ».

En programmation, vous pouvez créer de telles séquences d'instructions nommées. Certains langages de programmation les appellent fonctions ou sous-programmes. Dans App Inventor, elles sont appelées procédures. Une procédure est une séquence nommée de blocs que vous pouvez appeler depuis n'importe quel endroit dans une application.

La figure 21-11 est un exemple de procédure qui estime la distance, en miles, entre deux coordonnées GPS que vous lui envoyez.

¹ Ces blocs sont affichés avec des entrées en ligne, ce qui réduit la largeur des blocs. Vous pouvez cliquer avec le bouton droit sur les blocs pour basculer entre les entrées « Inline » et « Externe ».

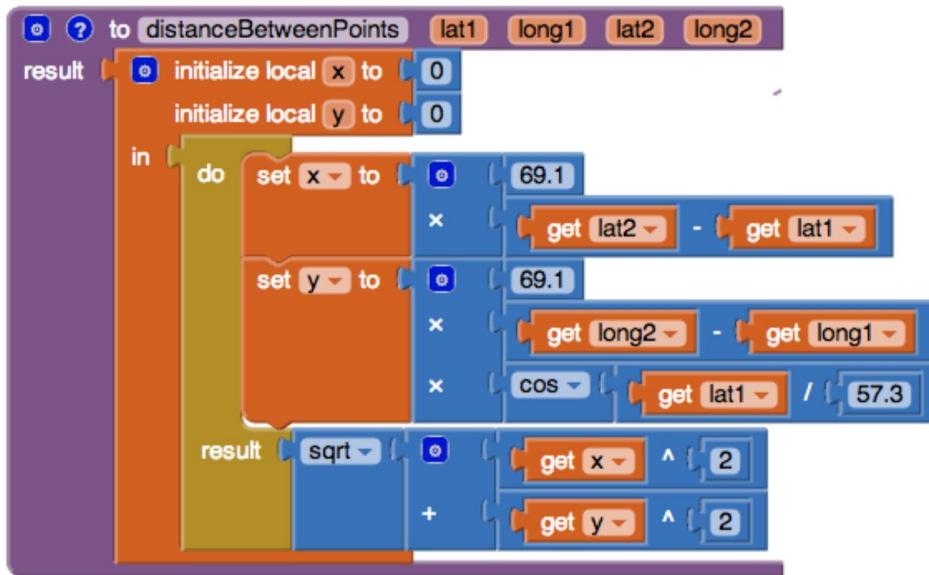


Figure 21-2. Procédure de calcul de la distance entre les points

Ne vous inquiétez pas trop des aspects internes de cette procédure pour l'instant ; tout ce dont tu as besoin Ce que je réalise actuellement, c'est que des procédures comme celle-ci vous permettent d'étendre le langage avec lequel vous concevez et construisez des programmes. Si chaque parent devait expliquer chaque soir les étapes à suivre pour « se brosser les dents » à son enfant, cet enfant n'atteindrait peut-être pas la cinquième année. Il est beaucoup plus efficace de simplement dire « Brossez-vous les dents » et tout le monde peut se coucher à une heure raisonnable.

De même, après avoir défini la procédure `distanceBetweenPoints`, vous pouvez ignorer les détails de son fonctionnement et faites simplement référence à (appelez) le nom de la procédure lors de la conception ou du codage d'une application plus grande. Ce type d'abstraction est essentiel pour résoudre de gros problèmes et nous permet de décomposer un grand projet logiciel en morceaux de code plus gérables.

Les procédures contribuent également à réduire les erreurs car elles éliminent la redondance dans votre code. Avec les procédures, vous pouvez placer un morceau de code au même endroit, puis l'appeler à partir de différents endroits de votre application. Ainsi, si vous créez une application qui doit connaître la distance minimale entre votre position actuelle et 10 autres points, vous n'avez pas besoin de 10 copies des blocs illustrés dans la figure 21-1. Au lieu de cela, vous définissez simplement la procédure `distanceBetweenPoints`, puis vous l'appelez chaque fois que vous en avez besoin. L'alternative – copier et coller des blocs – dépend beaucoup plus du code (rappelez-vous la discussion du chapitre 19) et, par conséquent, est sujette aux erreurs car lorsque vous effectuez une modification, vous devez rechercher toutes les autres copies de ces blocs et les modifier chacune. de la même manière. Imaginez que vous essayez de trouver les 5 à 10 endroits où vous avez collé un

morceau de code particulier dans une application avec 1 000 lignes ou blocs ! Une procédure vous permet d'encapsuler des blocs au même endroit, puis de l'appeler plusieurs fois.

Les procédures vous aident également à créer une bibliothèque de code pouvant être réutilisée dans de nombreux domaines. applications. Même lorsqu'ils créent une application dans un but très spécifique, les programmeurs expérimentés réfléchissent toujours à des moyens de créer le code de manière à pouvoir le réutiliser dans d'autres applications. Certains programmeurs ne créent même jamais d'applications, mais se concentrent uniquement sur la création de bibliothèques de code réutilisables que d'autres programmeurs pourront utiliser dans leurs applications !

Éliminer la redondance

Les blocs de la figure 21-2 proviennent d'une application Note Taker. Jetez un œil aux blocs et voyez si vous pouvez identifier ceux qui sont redondants.

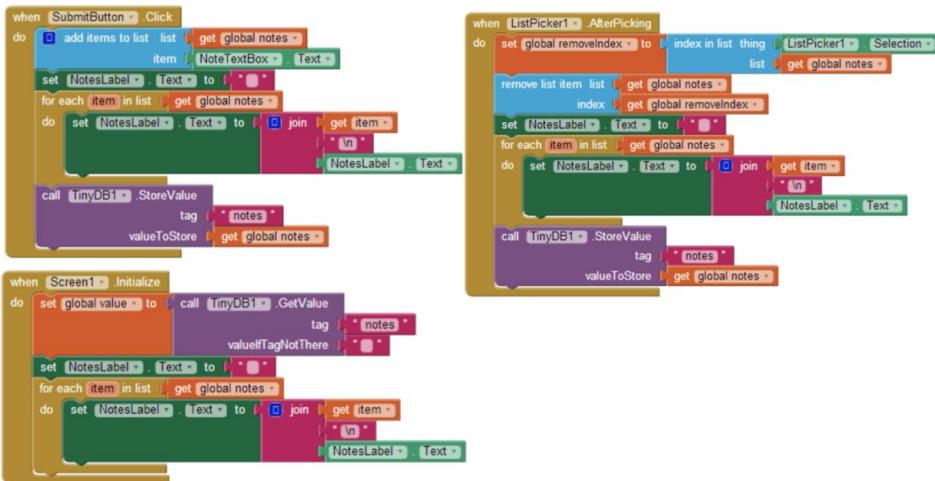


Figure 21-3. Une application Note Taker avec du code redondant

Les blocs redondants sont ceux impliquant un pour chaque bloc (en fait le pour chacun et ses blocs imbriqués et l'ensemble NotesLabel.Text au-dessus). Dans les trois cas, le travail du bloc est d'afficher la liste des notes. Dans cette application, ce comportement doit avoir lieu dans trois gestionnaires d'événements : lorsqu'un nouvel élément est ajouté, lorsqu'un élément est supprimé et lorsque la liste est chargée à partir de la base de données au lancement de l'application.

Lorsque des programmeurs expérimentés constatent une telle redondance, une cloche sonne dans leur tête, probablement avant même qu'ils n'aient copié et collé les blocs en premier lieu.

Ils savent qu'il est préférable d'encapsuler cette redondance dans une procédure, à la fois pour

rendre le programme plus compréhensible et pour que les modifications soient beaucoup plus faciles à apporter par la suite.

En conséquence, un programmeur expérimenté créerait une procédure, déplacerait une copie des blocs redondants, puis appellez la procédure depuis les trois emplacements contenant les blocs redondants. L'application ne se comportera pas différemment, mais elle sera plus facile à maintenir et à utiliser avec d'autres programmeurs. Une telle réorganisation du code (bloc) est appelée refactoring.

Définir une procédure

Construisons une procédure pour effectuer le travail des blocs de code redondants de la figure 21-2. Dans App Inventor, vous définissez une procédure de la même manière que vous définissez des variables.

Depuis le tiroir Procédures, faites glisser un bloc vers la procédure ou un bloc vers le résultat de la procédure . Utilisez cette dernière si votre procédure doit calculer une valeur et la renvoyer (nous discuterons de cette approche un peu plus tard dans le chapitre).

Après avoir fait glisser un bloc de procédure , vous pouvez modifier son nom par défaut en cliquant sur le mot « procédure » et en tapant un nouveau nom. Les blocs redondants que vous souhaitez refactoriser effectuent le travail d'affichage d'une liste, nous appellerons donc la procédure displayList, illustrée à la figure 21-3.



Figure 21-4. Cliquez sur « procédure » pour nommer votre procédure

L'étape suivante consiste à ajouter les blocs dans la procédure. Dans ce cas, nous utilisons des blocs qui existent déjà, nous allons donc faire glisser l'un des blocs redondants d'origine hors de son gestionnaire d'événements et le placer dans le bloc `to displayList` , comme le montre la figure 21-4.

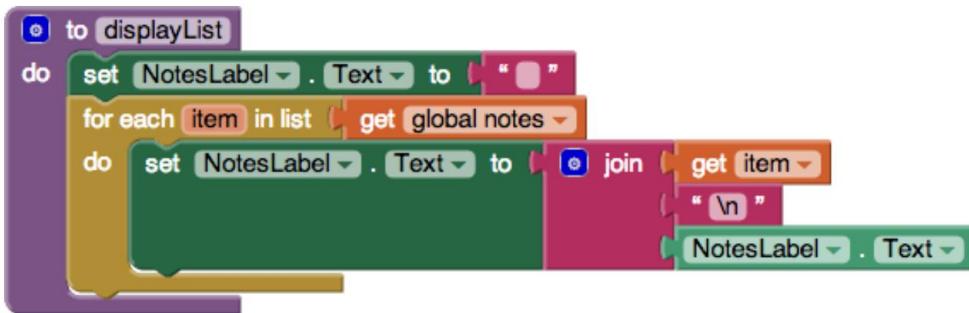


Figure 21-5. La procédure displayList encapsule le code redondant

Appel d'une procédure

Les procédures, comme displayList et « brossez-vous les dents », sont des entités ayant le potentiel d'effectuer une tâche. Cependant, ils n'accompliront cette tâche que s'ils sont appelés à le faire. Jusqu'à présent, nous avons créé une procédure mais ne l'avons pas appelée. Appeler une procédure signifie l'invoquer ou la réaliser.

Dans App Inventor, lorsque vous définissez une procédure, un bloc d'appel est automatiquement ajouté dans le tiroir Procédures, comme illustré à la Figure 21-6.

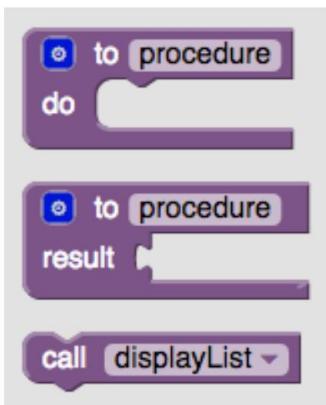


Figure 21-6. Un bloc d'appel apparaît dans le tiroir Procédures lorsque vous définissez une procédure

Vous avez déjà utilisé des blocs d'appel pour appeler les fonctions prédéfinies d'App Inventor, tels que Ball.MoveTo et Texting.SendMessage. Lorsque vous définissez une procédure, vous avez essentiellement créé votre propre bloc ; vous avez étendu le langage App Inventor.

À l'aide du nouveau bloc d'appel, vous pouvez invoquer votre création.

Pour l'exemple de l'application Note Taker, vous devez faire glisser trois blocs d'appel displayList et les utiliser pour remplacer le code redondant dans les trois gestionnaires d'événements. Par exemple, le

Le gestionnaire d'événements ListPicker1.AfterPicking (pour supprimer une note) doit être modifié comme illustré dans la figure 21-6.



Figure 21-7. Utilisation de l'appel displayList pour appeler les blocs maintenant dans la procédure

Le compteur de programmes

Pour comprendre le fonctionnement du bloc d'appel , imaginez une application comme un pointeur qui parcourt les blocs qui exécutent des fonctions. En informatique, ce pointeur est appelé compteur de programme.

Lorsque le compteur de programme exécute les blocs dans un gestionnaire d'événements et qu'il atteint un bloc d'appel , il passe à la procédure et exécute les blocs qu'elle contient.

Une fois la procédure terminée, le compteur de programme revient à son emplacement précédent (le bloc d'appel) et continue à partir de là. Ainsi, pour l'exemple de Note Taker, le bloc de suppression d'élément de liste est exécuté ; puis le compteur du programme passe à la procédure displayList et exécute les blocs de cette procédure (en définissant NotesLabel.Text sur le texte vide et pour chacun) ; et enfin le compteur du programme revient pour exécuter le bloc TinyDB1.StoreValue .

Ajout de paramètres à votre procédure

La procédure displayList permet de refactoiriser le code redondant en un seul endroit. L'application est plus facile à comprendre car vous pouvez lire les gestionnaires d'événements à un niveau élevé et ignorer généralement les détails de l'affichage d'une liste. C'est également utile car vous pouvez décider de modifier la façon dont vous affichez la liste, et la procédure vous permet d'effectuer une telle modification en un seul endroit (au lieu de trois).

La procédure displayList a cependant des limites en termes d'utilité générale.

La procédure ne fonctionne que pour une liste spécifique (notes) et affiche cette liste dans une étiquette spécifique (NotesLabel). Vous ne pouvez pas l'utiliser pour afficher une autre liste de données, par exemple une liste des utilisateurs de l'application, car elle est définie de manière trop spécifique.

App Inventor et d'autres langages fournissent un mécanisme appelé paramètres permettant de rendre les procédures plus générales. Les paramètres comprennent les informations dont une procédure a besoin pour faire son travail. Ils fournissent les détails de la manière dont la procédure doit être effectuée. Dans notre exemple de brossage des dents au coucher, vous pouvez définir le « type de dentifrice » et le « temps de brossage » comme paramètres de la procédure « se brosser les dents ».

Vous définissez les paramètres d'une procédure en cliquant sur l'icône bleue en haut à gauche de la définition de la procédure. Pour la procédure `displayList`, nous définirions un paramètre nommé « `list` », comme le montre la figure 21-7.

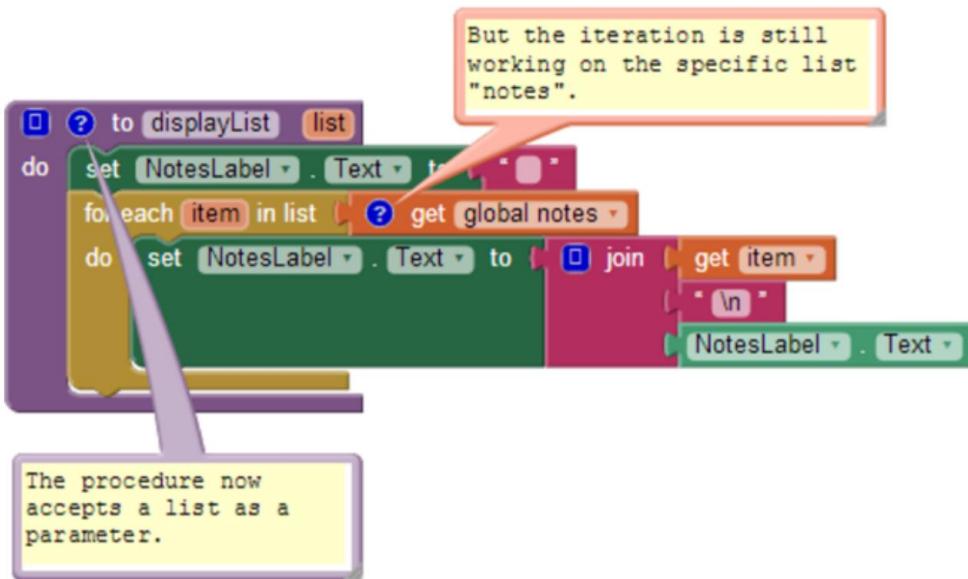


Figure 21-8. La procédure accepte désormais une liste en paramètre

Même avec le paramètre défini, les blocs font toujours directement référence à la liste spécifique `notes` (il est branché dans le slot « `in list` » du `for each`). Parce que nous voulons que la procédure utilise la liste que nous envoyons comme paramètre, nous remplaçons la référence aux notes globales par une référence pour obtenir la liste, comme le montre la figure 21-8.

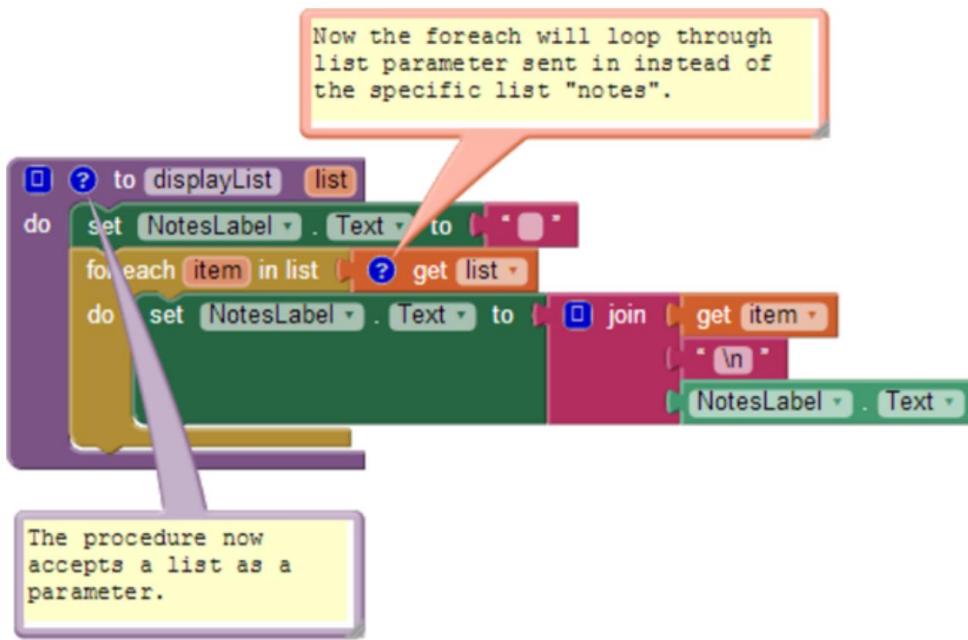
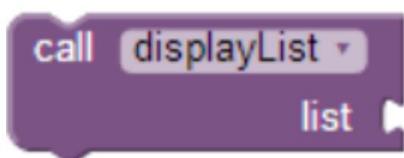


Figure 21-9. Maintenant, le for each utilisera la liste envoyée

La nouvelle version de la procédure est plus générique : les appels à `displayList` peuvent désormais envoyer-lui n'importe quelle liste et `displayList` l'affichera. Lorsque vous ajoutez un paramètre à une procédure, App Inventor place automatiquement un socket correspondant dans le bloc d'appel. Ainsi, lorsque la liste de paramètres est ajoutée à `displayList`, les blocs d'appel à `displayList` ressemblent à la figure 21-9.

Figure 21-10. L'appel de `displayList` vous oblige maintenant à spécifier quelle liste afficher

La liste de paramètres dans la définition de la procédure est appelée paramètre formel. Le socket correspondant dans le bloc d'appel est appelé un paramètre réel. Lorsque vous appelez une procédure depuis quelque part dans l'application, vous devez fournir un paramètre réel pour chaque paramètre formel de la procédure. Pour ce faire, remplissez toutes les sockets de l'appel.

Pour l'application Note Taker, vous ajoutez une référence pour obtenir des notes globales comme paramètre réel. La figure 21-10 montre comment `ListPicker.AfterSelection` doit être modifié.



Figure 21-11. Appel de `displayList` avec des notes envoyées comme paramètre réel

Désormais, lorsque `displayList` est appelé, les notes de la liste sont envoyées à la procédure et placées dans la liste des paramètres. Le compteur de programme procède à l'exécution des blocs de la procédure, en se référant à la liste des paramètres mais en travaillant réellement avec la variable

Remarques.

Grâce à ce paramètre, vous pouvez désormais utiliser la procédure `displayList` avec n'importe quel liste, pas seulement des notes. Par exemple, si l'application Note Taker était partagée entre une liste d'utilisateurs et que vous souhaitiez afficher la liste des utilisateurs, vous pouvez appeler `displayList` et lui envoyer la `userList`, comme illustré dans la figure 21-12.



Figure 21-12. La procédure `displayList` peut désormais être utilisée pour afficher n'importe quelle liste, pas seulement des notes

Renvoyer les valeurs d'une procédure

Il y a encore un problème avec la procédure `displayList` en termes d'utilité générale : pouvez-vous comprendre de quoi il s'agit ? Tel qu'il est actuellement écrit, il peut afficher n'importe quelle liste de données, mais il affichera toujours ces données dans l'étiquette `NotesLabel`. Que se passe-t-il si vous souhaitez que la liste soit affichée dans un objet d'interface utilisateur différent (par exemple, vous aviez une étiquette différente pour afficher la liste d'utilisateurs) ?

Une solution consiste à reconceptualiser la procédure et à modifier son travail, passant de l'affichage d'une liste dans une étiquette particulière au simple renvoi d'un objet texte que vous pouvez

afficher n'importe où. Pour ce faire, vous utilisez un bloc de résultat de procédure , illustré dans la figure 21-12, au lieu du bloc de procédure .



Figure 21-13. Le bloc résultat de la procédure

Vous remarquerez que, comparé au bloc de procédure , le résultat de la procédure Le bloc a une prise supplémentaire en bas. Vous placez une variable dans cet emplacement et elle est renvoyée à l'appelant. Ainsi, tout comme l'appelant peut envoyer des données à une procédure avec un paramètre, une procédure peut renvoyer des données avec une valeur de retour.

La figure 21-13 montre la version retravaillée de la procédure précédente, utilisant cette fois un bloc de résultat de procédure . Notez que, comme la procédure effectue désormais un travail différent, son nom passe de displayList à listToText.

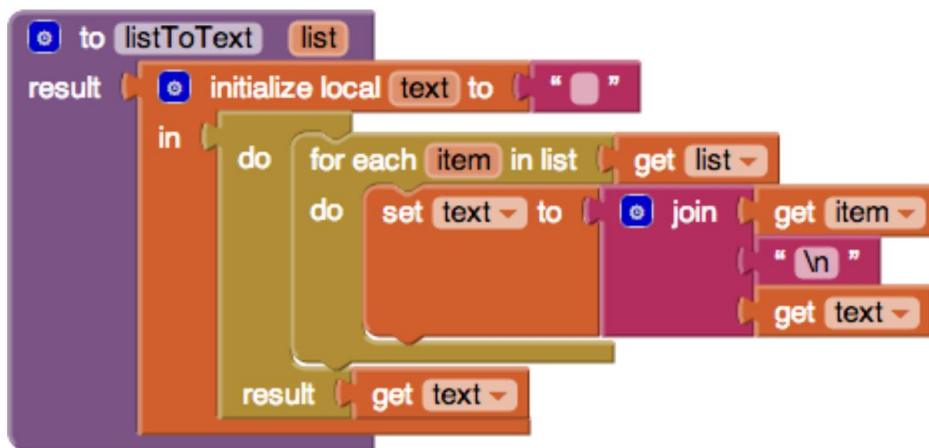


Figure 21-14. listToText renvoie un objet texte que l'appelant peut placer dans n'importe quelle étiquette

Dans les blocs illustrés à la figure 21-13, un texte de variable locale est défini pour contenir les données. à mesure que la procédure parcourt chaque élément de la liste. text est initialisé en tant que variable locale, au lieu de variable globale, car elle n'est utilisée que dans cette procédure.

Cette variable de texte remplace le composant NotesLabel trop spécifique qui était étant utilisé dans la version displayList de cette procédure. Lorsque le for each est terminé, la variable text contient les éléments de la liste, chaque élément étant séparé par un caractère de nouvelle ligne, \n (par exemple, « item1\nitem2\nitem3 »). Cette variable de texte est ensuite branchée sur le socket de valeur de retour.

Lorsqu'un résultat de procédure est défini, ses blocs d'appel correspondants sont différents que ceux d'une procédure. Comparez l'appel à `listToText` avec l'appel à `displayList` dans la figure 21-14.



Figure 21-15. L'appel à droite renvoie une valeur et doit donc être branché sur quelque chose

La différence est que l'appel `listToText` a une fiche sur son côté gauche. En effet, lorsque l'appel est exécuté, la procédure exécutera sa tâche puis renverra une valeur au bloc d'appel. Cette valeur de retour doit être connectée à quelque chose.

Dans ce cas, les appels de `displayList` peuvent insérer cette valeur de retour dans n'importe quelle étiquette de leur choix. Pour l'exemple de Note Taker, les trois gestionnaires d'événements qui doivent afficher une liste appelleront la procédure, comme le montre la figure 21-15.



Figure 21-16. Conversion des notes de la liste en texte et affichage dans NotesLabel

Le point important ici est que, comme la procédure est complètement générique et ne fait référence à aucune liste ou étiquette spécifiquement, une autre partie de l'application pourrait l'utiliser pour afficher n'importe quelle liste dans n'importe quelle étiquette, comme illustré dans la figure 21-16.



Figure 21-17. La procédure n'est plus liée à un composant Label particulier

Réutiliser des blocs entre les applications

La réutilisation de blocs de code via des procédures ne doit pas nécessairement être limitée à une seule application. Il existe de nombreuses procédures, telles que `listToText`, que vous pouvez utiliser dans presque toutes les applications que vous créez. Dans la pratique, les organisations et les communautés de programmation constituent des bibliothèques de codes de procédures pour leurs domaines d'intérêt.

En règle générale, les langages de programmation fournissent un utilitaire d'importation grâce auquel vous pouvez inclure du code de bibliothèque dans n'importe quelle application. App Inventor ne dispose pas encore d'un tel utilitaire. La seule façon de partager des procédures est de créer une application de bibliothèque spéciale et de commencer le développement d'une nouvelle application en enregistrant une nouvelle copie de cette application et en travaillant à partir de celle-ci.

La procédure distanceBetweenPoints

Avec l'exemple `displayList (listToText)`, nous avons caractérisé la définition de procédure comme un moyen d'éliminer le code redondant : vous commencez à écrire du code, trouvez les redondances au fur et à mesure et refactorisez votre code pour les éliminer. Cependant, en règle générale, un développeur de logiciels ou une équipe concevra une application dès le début en gardant à l'esprit les procédures et les pièces réutilisables. Ce type de planification peut vous faire gagner beaucoup de temps à mesure que le projet progresse.

Envisagez une application pour déterminer l'hôpital local le plus proche de l'emplacement actuel de l'utilisateur, ce qui serait très utile en cas d'urgence. Voici une description générale de la conception de l'application :

Lorsque l'application démarre, recherchez la distance, en miles, entre l'emplacement actuel et le premier hôpital. Trouvez-le ensuite pour le deuxième hôpital, et ainsi de suite. Lorsque vous avez les distances, déterminez la distance minimale et affichez l'adresse (et/ou une carte) de cet endroit.

À partir de cette description, pouvez-vous déterminer les procédures dont cette application a besoin ?

Souvent, les verbes dans une telle description font allusion aux procédures dont vous aurez besoin. Répétition dans votre description, comme indiqué par le « ainsi de suite », est un autre indice. Dans ce cas, trouver la distance entre deux points et déterminer le minimum de certaines distances sont deux procédures nécessaires.

Pensons à la conception de la procédure permettant de trouver la distance entre deux points, que nous appellerons `distanceBetweenPoints` (fne, donc l'originalité n'est pas mon point fort). Lors de la conception d'une procédure, vous devez déterminer ses entrées et sorties : les paramètres que l'appelant enverra à la procédure pour qu'elle fasse son travail, et la valeur du résultat que la procédure renverra à l'appelant. Dans ce cas, l'appelant doit envoyer la latitude et la longitude des deux points à la procédure, comme le montre la figure 21-17.

Le travail de la procédure consiste à renvoyer la distance, en miles.

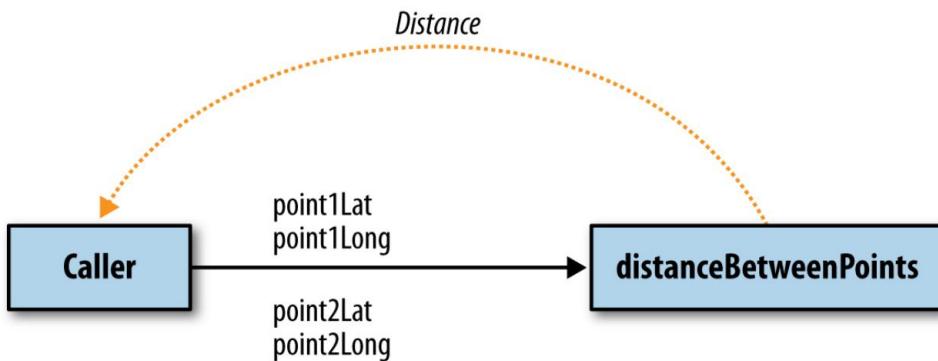


Figure 21-18. L'appelant envoie quatre paramètres d'entrée et reçoit une distance

La figure 21-18 montre la procédure que nous avons rencontrée au début du chapitre, en utilisant une formule pour approximer le kilométrage entre deux coordonnées GPS.

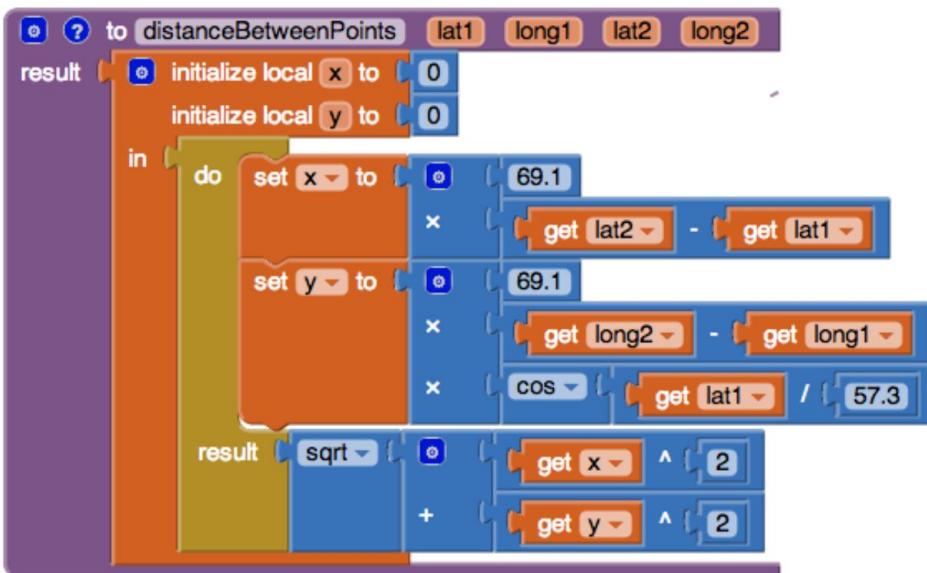


Figure 21-19. procédure distanceBetweenPoints

La figure 21-19 montre les blocs qui effectuent deux appels à la procédure, dont chacun trouve la distance entre l'emplacement actuel et un hôpital particulier.

Pour le premier appel, les paramètres réels du premier point sont les lectures actuelles du LocationSensor, tandis que le deuxième point est défini par les coordonnées GPS de l'hôpital St. Mary. La valeur résultante est placée dans la variable **distanceStMarys**. Le deuxième appel est similaire mais utilise à la place les données de l'hôpital CPMC pour le deuxième point.

L'application compare ensuite les deux distances renvoyées pour déterminer quel hôpital est le plus proche. Mais s'il y avait plus d'hôpitaux impliqués, il faudrait vraiment comparer une liste de distances pour trouver la plus courte. D'après ce que vous avez appris, pouvez-vous créer une procédure appelée `findMinimum` qui accepte une liste de nombres comme paramètre et renvoie l'index du minimum ?

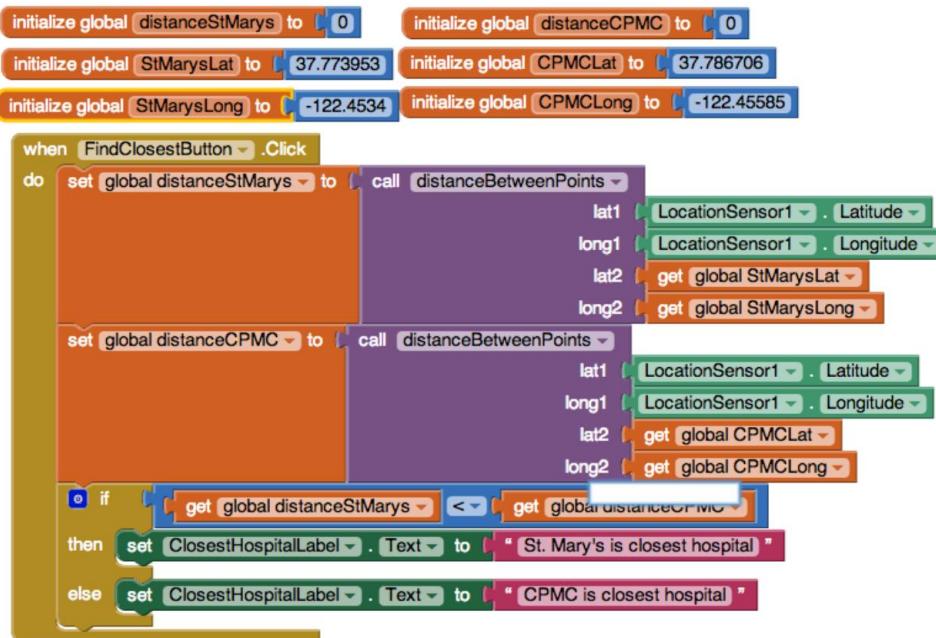


Figure 21-20. Deux appels à la procédure `distanceBetweenPoints`

Résumé

Les langages de programmation tels qu'App Inventor fournissent un ensemble de fonctionnalités de base intégrées. Grâce à l'utilisation de procédures, les inventeurs d'applications peuvent étendre ce langage avec de nouvelles abstractions. App Inventor ne fournit pas de bloc pour afficher une liste, vous en créez donc une.

Besoin d'un bloc pour calculer la distance entre les coordonnées GPS ?

Vous pouvez créer le vôtre.

La capacité à définir des blocs de procédures de niveau supérieur est la clé de l'ingénierie de grande envergure. Logiciel maintenable et résoudre des problèmes complexes sans être constamment submergé par tous les détails. Les procédures vous permettent d'encapsuler des blocs de code et de donner un nom à ces blocs. Pendant que vous programmez la procédure, vous vous concentrez uniquement sur les détails de ces blocs. Cependant, en programmant le reste de l'application, vous disposez désormais d'une abstraction – un nom – à laquelle vous pouvez vous référer à un niveau élevé.

Travailler avec des bases de données

Figure 22-1.



Facebook a une base de données de chaque membre

informations sur le compte, liste d'amis et publications.

Amazon dispose d'une base de données contenant à peu près tout ce que vous pouvez acheter. Google dispose d'une base de données d'informations sur chaque page du World Wide Web. Bien que pas à une telle échelle, presque toutes les applications non triviales que vous pouvez créer interagiront avec une base de données.

Dans la plupart des environnements de programmation, créer une application qui communique avec une base de données est une technique de programmation avancée : vous devez configurer un

serveur avec un logiciel de base de données tel qu'Oracle ou MySQL, puis écrire du code qui s'interface avec cette base de données. Dans de nombreuses universités, cette programmation de bases de données n'est enseignée qu'après un cours de génie logiciel ou de bases de données de niveau supérieur.

Lorsqu'il s'agit de bases de données, App Inventor fait le gros du travail à votre place (et bien d'autres encore). d'autres choses utiles !). Le langage fournit des composants qui réduisent la communication avec la base de données à de simples opérations de stockage et d'obtention. Vous pouvez créer des applications qui stockent des données directement sur l'appareil Android et, avec une certaine configuration, vous pouvez créer des applications qui partagent des données avec d'autres appareils et personnes en les stockant dans une base de données centralisée sur le Web.

Les données stockées dans les variables et les propriétés des composants sont à court terme : si l'utilisateur saisit des informations dans un formulaire puis ferme l'application avant que ces informations n'aient été stockées dans une base de données, les informations disparaîtront à la réouverture de l'application. Pour stocker des informations de manière persistante, vous devez les stocker dans une base de données. Les informations contenues dans les bases de données sont dites persistantes car même lorsque vous fermez l'application et la rouvrez, les données sont toujours disponibles.

À titre d'exemple, considérons l'application No Texting While Driving du chapitre 4, qui envoie une réponse automatique aux messages texte SMS entrants. L'application a une réponse par défaut qui est envoyée, mais elle permet à l'utilisateur de saisir un message personnalisé à envoyer. Si l'utilisateur modifie le message personnalisé en « Je dors ; arrête de m'embêter » et ferme ensuite l'application, le message devrait toujours être « Je dors ; arrête de me déranger », et non la valeur par défaut d'origine, lorsque l'application est rouverte. Ainsi, le message personnalisé doit être stocké dans une base de données et chaque fois que l'application est ouverte, ce message doit être récupéré de la base de données dans l'application.

Stockage de données persistantes dans TinyDB

App Inventor fournit deux composants pour faciliter l'activité de la base de données : TinyDB et TinyWebDB. Vous utilisez TinyDB pour stocker des données persistantes directement sur l'appareil Android ; ceci est utile pour les applications personnelles pour lesquelles l'utilisateur n'aura pas besoin de partager des données avec un autre appareil ou une autre personne, comme dans Pas d'envoi de SMS pendant la conduite. D'un autre côté, vous utilisez TinyWebDB pour stocker des données dans une base de données Web pouvant être partagée entre les appareils. Pouvoir accéder aux données d'une base de données Web est essentiel pour les jeux et applications multi-utilisateurs avec lesquels les utilisateurs peuvent saisir et partager des informations (comme l'application « MakeQuiz » au chapitre 10).

Les composants de la base de données sont similaires, mais TinyDB est un peu plus simple, nous allons donc l'explorer premier. Avec TinyDB, vous n'avez pas du tout besoin de configurer la base de données ; les données sont stockées dans une base de données directement sur l'appareil et associées à votre application.

Vous transférez des données vers la mémoire à long terme avec le bloc TinyDB.StoreValue , comme illustré à la figure 22-1, qui provient de l'application No Texting While Driving.

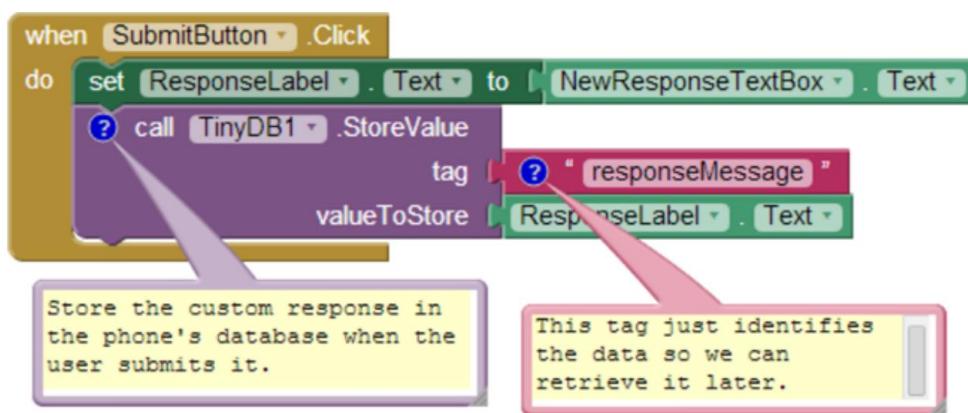


Figure 22-2. Le bloc TinyDB.StoreValue stocke les données dans la mémoire à long terme de l'appareil

Un schéma de valeur de balise est utilisé pour le stockage de la base de données. Dans la figure 22-1, les données sont étiquetées avec le texte «responseMessage». La valeur est un texte que l'utilisateur a tapé dans une zone de texte pour la nouvelle réponse personnalisée, par exemple : « Je dors ; arrêter de me casser les pieds.»

Le paramètre tag donne un nom aux données que vous stockez dans la base de données, un moyen de référencer les informations. La valeur est la donnée elle-même. Vous pouvez considérer la balise comme une clé que vous utiliserez plus tard lorsque vous souhaiterez récupérer les données de la base de données.

De même, vous pouvez considérer une base de données App Inventor TinyDB comme une table de valeurs de balises. Paires. Une fois TinyDB1.StoreValue de la figure 22-1 exécuté, la base de données du périphérique aura la valeur répertoriée dans le tableau 22-1.

Tableau 22-1. La valeur stockée dans les bases de données

Étiqueter	Valeur
réponseMessage	Je dors ; arrêter de me casser les pieds

Une application peut stocker de nombreuses paires balise-valeur pour les différents éléments de données que vous souhaitez rendre persistants. La balise est toujours du texte, tandis que la valeur peut être soit une seule information (un texte ou un nombre), soit une liste. Chaque balise n'a qu'une seule valeur ; chaque fois que vous stockez dans une balise, la valeur existante est écrasée.

Récupérer des données de TinyDB

Vous récupérez les données de la base de données à l'aide du bloc TinyDB.GetValue . Lorsque vous appelez GetValue, vous demandez des données particulières en fournissant une balise. Pour l'application No Texting While Driving, vous pouvez demander la réponse personnalisée en utilisant la même balise que celle que vous avez utilisée dans StoreValue , «responseMessage». L'appel à GetValue renvoie les données, vous devez donc les brancher dans une variable.

Souvent, vous récupérez les données de la base de données à l'ouverture de l'application. App Inventor fournit un gestionnaire d'événements spéciaux, Screen.Initialize, qui est déclenché au lancement de l'application. Vous devez faire attention au cas où il n'y a pas encore de données dans la base de données (par exemple, la première fois que l'application est lancée). Lorsque vousappelez GetValue, vous spécifiez un paramètre valueIfTagNotThere . S'il n'y a pas de données, cette valeur sera renvoyée par l'appel.

Les blocs de la figure 22-2, pour l'écran. Initialisation de l'absence de SMS pendant la conduite app, indiquent la manière dont de nombreuses applications chargent les données de la base de données lors de l'initialisation. Les blocs placent les données renvoyées par GetValue dans l'étiquette ResponseLabel. Si des données existent déjà dans la base de données, elles sont placées dans ResponseLabel. S'il n'y a aucune donnée pour la balise donnée, la valeur valueIfTagNotThere , « Je conduis en ce moment, je vous enverrai un SMS plus tard » dans ce cas, est placée dans ResponseLabel.



Figure 22-3. Au lancement de l'application, vous récupérez souvent les informations de la base de données

Données partagées et TinyWebDB

Le composant TinyDB stocke les données dans une base de données située directement sur l'appareil Android.

Ceci est approprié pour les applications à usage personnel qui n'ont pas besoin de partager des données entre les utilisateurs. Par exemple, de nombreuses personnes peuvent installer l'application No Texting While Driving, mais les différentes personnes utilisant l'application n'ont pas besoin de partager leurs réponses personnalisées avec d'autres.

Bien sûr, de nombreuses applications partagent des données : pensez à Facebook, Twitter et aux jeux multi-utilisateurs. Pour de telles applications de partage de données, la base de données doit résider sur le Web, et non sur l'appareil, afin que différents utilisateurs de l'application puissent communiquer avec elle et accéder à ses informations.

TinyWebDB est l'équivalent Web de TinyDB. Avec lui, vous pouvez écrire des applications qui stockent données sur le Web, en utilisant un protocole StoreValue/GetValue similaire à celui de TinyDB.

Par défaut, le composant TinyWebDB stocke les données à l'aide d'une base de données Web configurée par l'équipe App Inventor et accessible sur <http://appinvintinywebdb.appspot.com>. Ce site Web contient une base de données et répond aux demandes Web de stockage et de récupération de données. Le site fournit également une interface Web lisible par l'homme qu'un administrateur de base de données (vous) peut utiliser pour examiner les données qui y sont stockées.

Cette base de données par défaut est uniquement destinée au développement ; il est de taille limitée et accessible à tous les programmeurs App Inventor. Étant donné que n'importe quelle application App Inventor peut y stocker des données, vous n'avez aucune garantie qu'une autre application n'écrasera pas vos données !

Si vous explorez simplement App Inventor ou si vous êtes aux premiers stades d'un projet, la base de données Web par défaut est fine. Mais si vous créez une application pour un déploiement réel, vous devrez à un moment donné configurer votre propre base de données Web. Parce que nous sommes en train d'explorer en ce moment, nous utiliserons la base de données Web par défaut. Plus loin dans le chapitre, vous apprendrez comment créer votre propre base de données Web et configurer TinyWebDB pour l'utiliser à la place.

Dans cette section, nous allons créer une application de vote (représentée dans la figure 22-3) pour illustrer le fonctionnement de TinyWebDB . L'application aura les fonctionnalités suivantes :

- Les utilisateurs sont invités à saisir leur adresse e-mail à chaque chargement de l'application. Que le nom du compte sera utilisé pour marquer le vote de l'utilisateur dans la base de données.
- Les utilisateurs peuvent soumettre un nouveau vote à tout moment. Dans ce cas, leur ancien vote sera écrasé.
- Les utilisateurs peuvent consulter les votes de tous les membres du groupe.
- Par souci de simplicité, le sujet soumis au vote est déterminé en dehors de l'application, par exemple dans une salle de classe dans laquelle l'enseignant annonce le sujet et demande à tout le monde de voter électroniquement. (Notez que cet exemple pourrait être étendu pour permettre aux utilisateurs de demander des votes en publiant des problèmes sur lesquels voter depuis l'application.)



Figure 22-4. Une application de vote qui stocke les votes dans TinyWebDB

STOCKAGE DE DONNÉES À L'AIDE DE TINYWEBDB

Le bloc TinyWebDB.StoreValue fonctionne de la même manière que TinyDB.StoreValue, sauf que les données sont stockées sur le Web. Pour notre exemple de vote, supposons que l'utilisateur puisse saisir un vote dans une zone de texte nommée VoteTextBox et appuyer sur un bouton nommé VoteButton pour soumettre le vote. Pour stocker le vote dans la base de données Web afin que d'autres

Si vous pouvez le voir, nous allons coder le gestionnaire d'événements VoteButton.Click comme dans l'exemple de la figure 22-4.

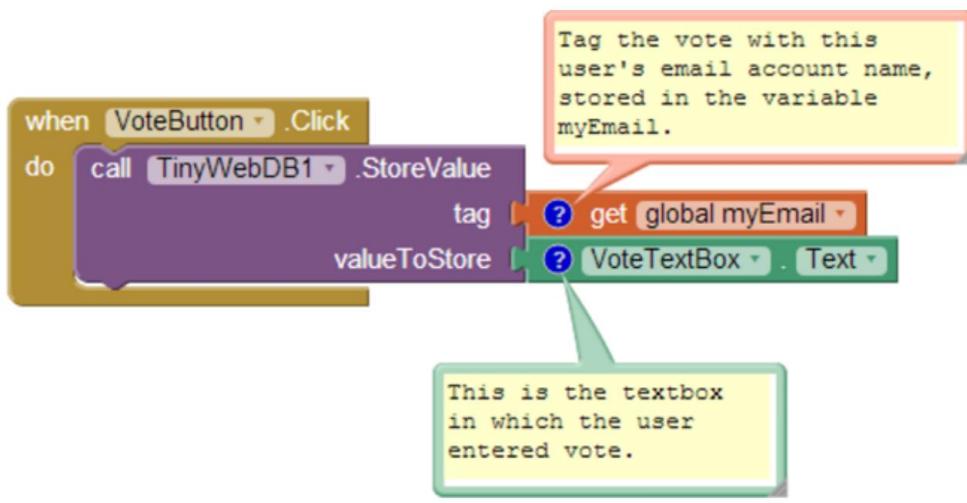


Figure 22-5. Lorsque l'utilisateur saisit un vote, celui-ci est stocké dans la base de données Web

La balise utilisée pour identifier les données est l'email de l'utilisateur, qui a été préalablement stocké dans la variable myEmail (vous le verrez plus tard). La valeur correspond à celle saisie par l'utilisateur dans VoteTextBox. Ainsi, si l'adresse e-mail de l'utilisateur était joe@zmail.com et que son vote était « Pizza », l'entrée serait stockée dans la base de données comme indiqué dans le tableau 22-2.

Tableau 22-2. Le tag et la valeur du vote sont enregistrés dans la base de données

étiqueter	valeur
joe@zmail.com	Pizza

Le bloc TinyWebDB.StoreValue envoie la paire balise-valeur via le Web au serveur de base de données à l' adresse <http://appinvtinywebdb.appspot.com>. Pendant que vous testez votre application, vous pouvez accéder à cette URL, cliquer sur getValue et saisir une balise pour laquelle vous avez stocké une valeur. Le site Web vous montrera la valeur actuelle de cette balise.

DEMANDE ET TRAITEMENT DE DONNÉES AVEC TINYWEBDB

Récupérer des données avec TinyWebDB est plus compliqué qu'avec TinyDB. Avec TinyDB, l' opération GetValue renvoie immédiatement une valeur car votre application communique avec une base de données directement sur l'appareil Android. Avec TinyWebDB, l'application demande des données sur le Web, ce qui peut prendre du temps. Android nécessite donc un schéma en deux étapes pour les gérer.

Avec TinyWebDB, un appel à GetValue demande uniquement les données ; ça devrait vraiment s'appeler "RequestValue" car il fait simplement la requête à la base de données Web et ne le fait pas

en tirer une valeur immédiatement . Pour voir cela plus clairement, vérifiez la différence entre le bloc TinyDB.GetValue et le bloc TinyWebDB.GetValue illustré dans la figure 22-5.



Figure 22-6. Les blocs TinyDB.GetValue et TinyDB.GotValue

Le bloc TinyDB.GetValue renvoie immédiatement une valeur, et ainsi un plug apparaît sur son côté gauche afin que la valeur renvoyée puisse être placée dans une variable ou une propriété. Le bloc TinyWebDB.GetValue ne renvoie pas de valeur immédiatement, il n'y a donc pas de plug sur son côté gauche.

Au lieu de cela, lorsque la base de données Web répond à la demande et que les données reviennent au périphérique, un événement TinyWebDB.GotValue est déclenché. Ainsi, vous appellerez TinyWebDB.GetValue à un endroit de votre application, puis vous programmerez le gestionnaire d'événements TinyWebDB.GotValue pour spécifier comment gérer les données lorsqu'elles arrivent réellement. Un gestionnaire d'événements tel que TinyWebDB.GotValue est parfois appelé procédure de rappel, car une entité externe (la base de données Web) rappelle effectivement votre application après avoir traité votre demande. C'est comme commander dans un café très fréquenté : vous passez votre commande, puis attendez que le barista vous appelle pour aller chercher votre boisson. En attendant, elle reçoit également les ordres de tous les autres (et ces gens attendent tous que leurs noms soient également appelés).

GETVALUE-GOTVALUE EN ACTION

Pour notre exemple d'application, nous devons stocker et récupérer une liste des électeurs qui possèdent l'application, car l'application doit afficher les votes de tous les utilisateurs.

Le schéma le plus simple pour récupérer les données d'une liste consiste à demander les données au lancement de l'application, dans l' événement Screen.Initialize , comme le montre la figure 22-6. (Dans cet exemple, nous appellerons simplement la base de données avec la balise « liste électorale ».)

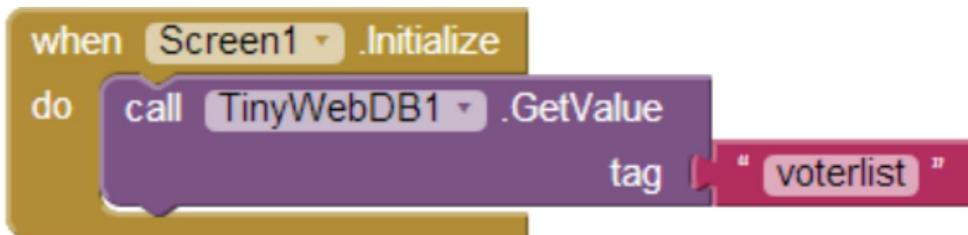


Figure 22-7. Demande de données dans l'événement Screen1.Initialize

Lorsque la liste des électeurs arrive de la base de données Web, le gestionnaire d'événements TinyWebDB1.GotValue est déclenché. La figure 22-7 montre quelques blocs pour traiter la liste renvoyée.

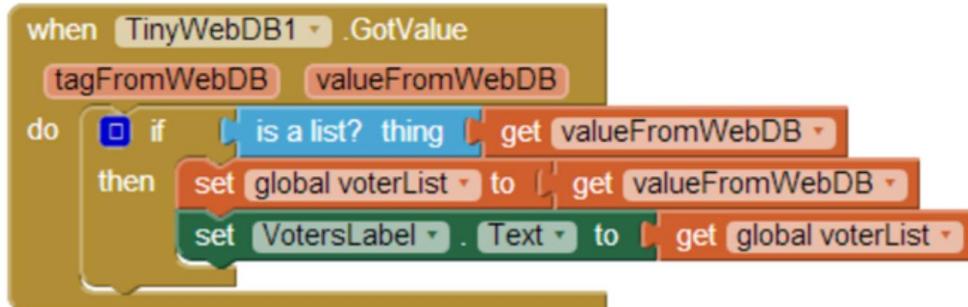


Figure 22-8. Utilisation du gestionnaire d'événements GotValue pour traiter la liste renvoyée

L'argument valueFromWebDB de GotValue contient les données renvoyées par la requête de base de données. Les arguments d'événement tels que valueFromWebDB n'ont de signification que dans le gestionnaire d'événements qui les appelle. Ils sont considérés comme locaux au gestionnaire d'événements, car vous ne pouvez pas les référencer dans d'autres gestionnaires d'événements.

Étant donné que les arguments tels que valueFromWebDB ne sont pas accessibles globalement, si vous avez besoin des informations dans votre application, vous devez les transférer vers une variable globale. Dans l'exemple, la tâche principale de GotValue consiste à transférer les données renvoyées dans valueFromWebDB dans la variable voterList, que vous utiliserez dans un autre gestionnaire d'événements.

Le bloc if dans le gestionnaire d'événements est également souvent utilisé en conjonction avec GotValue, la raison étant que la base de données renvoie un texte vide (« ») dans valueFromWebDB s'il n'y a pas de données pour la balise demandée. Cette valeur de retour vide se produit le plus souvent lors de la première utilisation de l'application. En demandant si valueFromWebDB est une liste, vous vous assurez que certaines données sont réellement renvoyées. Si valueFromWebDB est le texte vide (le test if est faux), vous ne le mettez pas dans voterList.

UN EXEMPLE GETVALUE/GOTVALUE PLUS COMPLEXE

Les blocs de la figure 22-7 constituent un bon modèle pour récupérer des données dans une application assez simpliste. Cependant, dans notre exemple de vote, nous avons besoin d'une logique plus compliquée. Plus précisément :

- L'application doit inviter l'utilisateur à saisir une adresse e-mail au démarrage du programme. Nous pouvons utiliser un composant Notifier pour cela, qui ouvre une fenêtre.
(Vous pouvez trouver le notificateur dans la palette « Interface utilisateur » du concepteur.)
Lorsque l'utilisateur saisit un e-mail, nous le stockons dans une variable.
- Ce n'est qu'après avoir déterminé l'e-mail de l'utilisateur que vous devez appeler GetValue pour récupérer la liste électorale. Peux-tu comprendre pourquoi?

La figure 22-8 montre les blocs de ce schéma plus compliqué de demande de données de base de données.

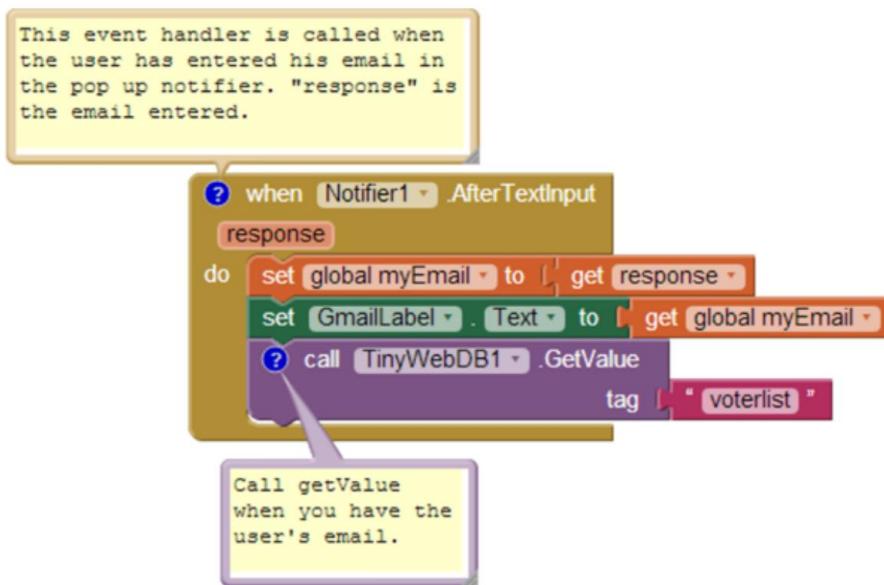


Figure 22-9. Dans ce schéma plus complexe, GetValue est appelé après avoir reçu l'e-mail de l'utilisateur plutôt que dans Screen.Initialize

Au démarrage (Screen1.Initialize), un composant Notifier invite l'utilisateur à saisir une adresse e-mail. Lorsque l'utilisateur le fait et que le gestionnaire d'événements Notifier.AfterTextInput est déclenché, l'entrée est placée dans une variable et une étiquette, puis GetValue est appelé pour obtenir la liste des votants. Notez que GetValue n'est pas appelé directement dans Screen.Initialize, car nous avons besoin que l'adresse e-mail de l'utilisateur soit définie en premier.

Ainsi, avec ces blocs, lorsque l'application s'initialise, elle invite l'utilisateur à saisir une adresse e-mail, puis appelle GetValue avec la balise « liste électorale ». Lorsque la liste arrive du Web, GotValue est déclenchée. Voici ce qui devrait se passer :

- GotValue doit vérifier si les données qui arrivent ne sont pas vides (quelqu'un a utilisé l'application et initié la liste électorale). S'il existe des données (une liste électorale), GotValue doit vérifier si l'adresse e-mail de notre utilisateur particulier figure déjà dans la liste électorale. Si ce n'est pas le cas, il doit être ajouté à la liste et la liste mise à jour doit être stockée dans la base de données.
- S'il n'y a pas encore de liste électorale dans la base de données, nous devrions en créer une avec le nom de l'utilisateur. adresse e-mail comme seul élément.

La figure 22-9 montre les blocs pour ce comportement.

Les blocs demandent d'abord si une liste électorale non vide est revenue de la base de données en appelant `est-ce une liste ?` Si tel est le cas, les données sont placées dans la variable `voterList`. N'oubliez pas que `voterList` aura les adresses e-mail de tous ceux qui ont utilisé cette application. Cependant, nous ne savons pas encore si cet utilisateur particulier figure dans la liste, nous devons donc vérifier. Si l'utilisateur n'est pas encore dans la liste, son adresse e-mail est ajoutée avec l'ajout d'un élément à la liste et la liste mise à jour est stockée dans la base de données Web.

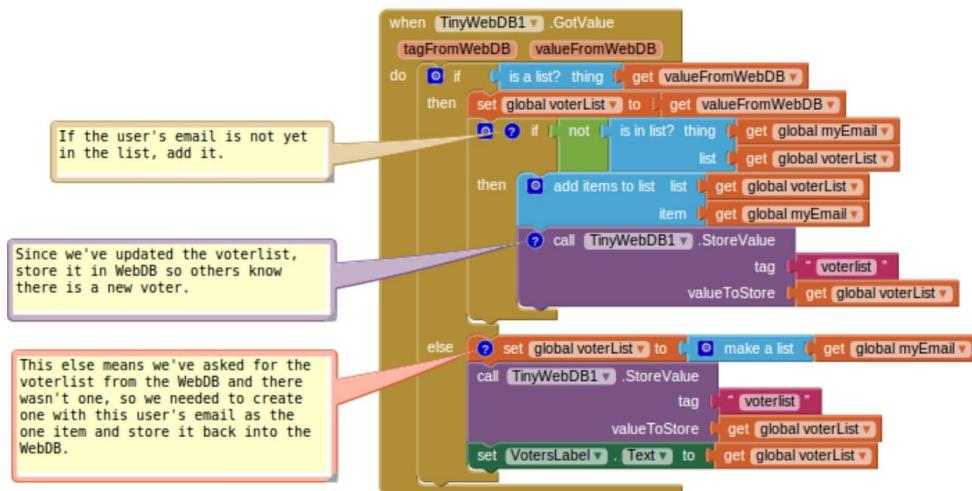


Figure 22-10. Utiliser les blocs `GetValue` pour traiter les données renvoyées par la base de données et effectuer différentes actions en fonction de ce qui est renvoyé

Le bloc `else` du bloc `if else` est invoqué si une liste n'a pas été renvoyée par le Web.

base de données; cela se produit si personne n'a encore utilisé l'application. Dans ce cas, une nouvelle liste d'électeurs est créée avec l'adresse e-mail de l'utilisateur actuel comme premier élément. Cet électeur à un seul élément

La liste est ensuite stockée dans la base de données Web (avec l'espoir que d'autres la rejoindront également !).

Demander des données avec diverses balises

Jusqu'à présent, l'application de vote gère une liste d'utilisateurs d'une application. Chaque personne peut voir les adresses email de tous les autres utilisateurs, mais nous n'avons pas encore créé de blocs pour récupérer et afficher le vote de chaque utilisateur.

Rappelez-vous que l'événement `VoteButton.Click` a soumis un vote avec une paire balise-valeur de le formulaire « email : votez ». Si deux personnes avaient utilisé l'application et voté, les entrées pertinentes de la base de données ressembleraient à quelque chose comme le tableau 22-3.

Tableau 22-3. Les paires balise-valeur stockées dans la base de données

équation	valeur
liste électorale	[bill@zmail.com, joe@zmail.com]
bill@zmail.com	Hot-dogs
joe@zmail.com	Pizza

Lorsque l'utilisateur clique sur le bouton ViewVotes , l'application doit récupérer tous les votes de la base de données et les afficher. Supposons que la liste électorale ait déjà été récupérée dans la variable voterList ; nous pouvons utiliser un for each pour demander le vote de chaque personne dans la liste, comme le montre la figure 22-10.

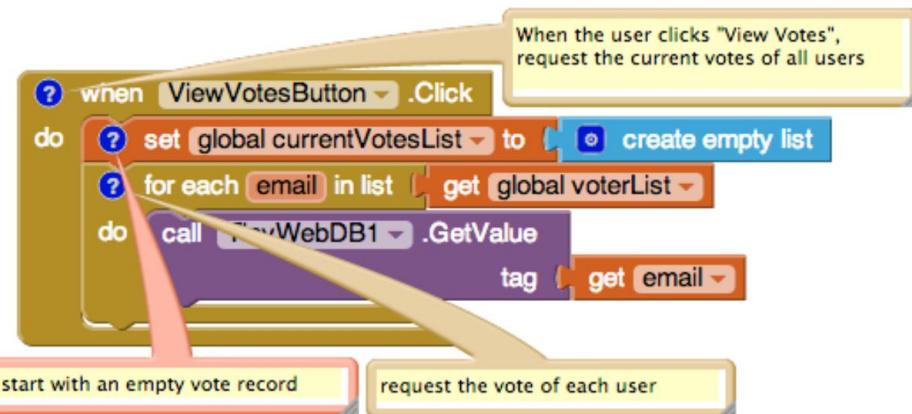


Figure 22-11. Utiliser un pour chaque bloc pour demander le vote de chaque personne de la liste

Ici, nous initialisons une variable, currentVotesList, sur une liste vide, car notre objectif est pour ajouter les votes à jour de la base de données dans cette liste. Nous utilisons ensuite for each pour appeler TinyWebDB1.GetValue pour chaque adresse e-mail de la liste, en envoyant l'élément actuel de for each, renommé « email », comme balise dans la requête. Notez que les votes ne seront pas réellement ajoutés à currentVotesList jusqu'à ce qu'ils arrivent via une série de GotValue événements.

Maintenant que nous souhaitons afficher les votes dans notre application, les choses se compliquent encore un peu. Avec les requêtes de ViewVotesButton, TinyWebDB.GetValue renverra désormais les données relatives à toutes les balises de messagerie, ainsi que la balise « voterlist » utilisée pour récupérer la liste des adresses e-mail des utilisateurs. Lorsque votre application demande plusieurs éléments de la base de données avec différentes balises, vous devez coder TinyWebDB.GetValue pour gérer toutes les demandes possibles. (Vous pourriez penser que vous pourriez essayer de coder plusieurs

Gestionnaires d'événements GotValue , un pour chaque requête de base de données : pouvez-vous comprendre pourquoi cela ne fonctionnera pas ?)

Pour gérer cette complexité, le gestionnaire d'événements GotValue dispose d'un argument tagFromWebDB qui vous informe de la requête qui vient d'arriver. Dans ce cas, si le tag est « liste électorale », nous devons continuer à traiter la demande comme nous le faisions précédemment. Si la balise est autre chose, nous pouvons supposer qu'il s'agit de l'e-mail d'une personne figurant dans la liste des utilisateurs, provenant des requêtes déclenchées dans le gestionnaire d'événements ViewVotesButton.Click . Lorsque ces demandes arrivent, nous souhaitons ajouter les données entrantes (l'électeur et le vote) à la currentVotesList afin de pouvoir les afficher à l'utilisateur.

La figure 22-11 montre l'intégralité du gestionnaire d'événements TinyWebDB.GotValue .

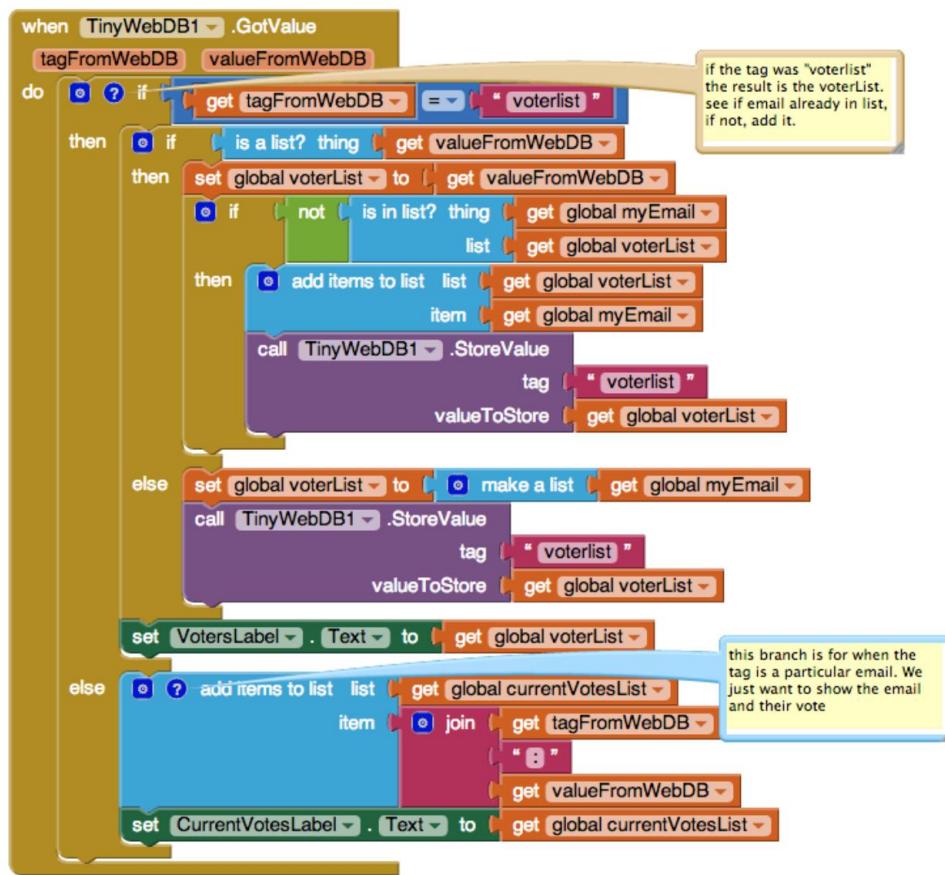


Figure 22-12. Le gestionnaire d'événements TinyWebDB.GotValue

Configuration d'une base de données Web

Comme nous l'avons mentionné plus tôt dans le chapitre, la base de données Web par défaut sur <http://appinvtinywebdb.appspot.com> est destinée uniquement à des fins de prototypage et de test.

Avant de déployer une application avec de vrais utilisateurs, vous devez créer une base de données spécifiquement pour votre application.

Vous pouvez créer une base de données Web en suivant les instructions sur <http://appinventorapi.com/create-a-web-database-python-2-7>. Ce site a été créé par l'un des auteurs (Wolber) et contient des exemples de code et des instructions pour configurer les bases de données Web et les API App Inventor. Les instructions vous indiquent du code que vous pouvez télécharger et utiliser avec seulement une modification mineure d'un fichier de configuration. Le téléchargement du code est le même que celui utilisé pour la base de données Web par défaut configurée par App Inventor. Il fonctionne sur App Engine de Google, un service de cloud computing qui hébergera gratuitement votre base de données Web sur les serveurs de Google (enfin, au moins jusqu'à ce que le site reçoive un certain nombre de visites). En suivant les instructions, vous pouvez disposer de votre propre base de données Web privée, conforme aux protocoles d'App Inventor, opérationnelle en quelques minutes et commencer à créer des applications mobiles Web qui l'utilisent.

Lorsque vous créez et déployez votre propre base de données Web personnalisée, l'outil App Engine vous fournit une URL où réside votre serveur. Vous pouvez demander à votre application d'utiliser votre serveur de base de données personnalisé au lieu du <http://appinvtinywebdb.appspot.com> par défaut , en modifiant la propriété ServiceURL dans le composant TinyWebDB . Une fois cette propriété modifiée, tous les appels à TinyWebDB.StoreValue et TinyWebDB.GetValue s'interfaceront avec le nouveau service Web.

Résumé

App Inventor facilite le stockage persistant des données via ses composants TinyDB et TinyWebDB . Les données sont toujours stockées sous forme de paire balise-valeur, la balise identifiant les données pour une récupération ultérieure. Utilisez TinyDB lorsqu'il est approprié de stocker des données directement sur l'appareil. Lorsque les données doivent être partagées entre téléphones (par exemple, pour un jeu multijoueur ou une application de vote), vous devrez plutôt utiliser TinyWebDB . TinyWebDB est plus compliqué car vous devez configurer une procédure de rappel (le gestionnaire d'événements GotValue) ainsi qu'un service de base de données Web.

Lire et répondre aux capteurs

Pointez votre téléphone vers le ciel et Google Sky Map vous indiquera les étoiles que vous regardez. Inclinez votre téléphone et vous pourrez contrôler le jeu auquel vous jouez. Emportez votre téléphone lors de votre course quotidienne et une application enregistre votre itinéraire. Toutes ces applications sont possibles car les appareils mobiles que nous transportons sont dotés de capteurs de haute technologie pour détecter notre emplacement, notre orientation et notre accélération.

Dans ce chapitre, vous explorerez les composants App Inventor LocationSensor, OrientationSensor et AccelerometerSensor. En chemin, vous en apprendrez davantage sur le système de positionnement global (GPS) ; mesures d'orientation telles que le tangage, le roulis et l'azimut ; et quelques mathématiques pour traiter les lectures de l'accéléromètre.

Créer une géolocalisation

applications

Jusqu'à la popularisation du smartphone, l'informatique était verrouillée sur les ordinateurs de bureau. Oui, les ordinateurs portables sont mobiles, mais pas dans le même sens que les petits appareils que nous transportons désormais dans nos poches. L'informatique a quitté le laboratoire et le bureau et s'étend désormais dans le monde entier, au-delà des contraintes des quatre murs.

L'un des effets importants du transport de notre ordinateur avec nous est une nouvelle fonctionnalité très intéressante. élément de données pour chaque application : un emplacement actuel. Savoir où se trouvent les gens lorsqu'ils se déplacent dans le monde a des implications considérables et peut potentiellement nous aider grandement dans nos vies. Cela a également le potentiel d'envahir notre vie privée et de nuire à l'humanité.

L'Android, où est ma voiture ? app (Chapitre 7) est un exemple d'application géolocalisée qui offre un avantage personnel. Il vous permet de mémoriser un emplacement précédent afin de pouvoir y revenir plus tard. Cette application est privée, ce qui signifie que vos informations de localisation sont stockées uniquement dans la base de données de votre appareil.

Figure 23-1.



348 Chapitre 23 : Lire et répondre aux capteurs

Les groupes peuvent également utiliser la détection de localisation. Par exemple, un groupe de randonneurs pourrait vouloir garder une trace des déplacements des uns et des autres dans la nature, ou un groupe d'associés pourrait vouloir se retrouver lors d'une grande conférence (ou d'un bar). Certaines personnes utilisent quotidiennement de telles applications d'enregistrement.

Un autre type d'application géolocalisée utilise des outils de réalité augmentée. Ces applications utilisent votre emplacement et l'orientation du téléphone pour fournir des informations superposées qui augmentent le cadre naturel. Ainsi, vous pouvez pointer un téléphone vers un immeuble et voir son prix sur le marché immobilier, ou vous pouvez vous promener près d'une plante exotique dans un jardin botanique et une application peut vous indiquer son espèce.

LE SYSTÈME DE POSITIONNEMENT GLOBAL

Pour créer une application de localisation, vous devez d'abord comprendre le fonctionnement du système de positionnement global (GPS). Les données GPS sont générées via une série de satellites géosynchrones gérés par le gouvernement des États-Unis. Tant que vous disposez d'une ligne de vue dégagée vers au moins trois satellites du système, votre téléphone peut obtenir une lecture. Une lecture GPS comprend votre latitude, votre longitude et votre altitude. La latitude correspond à votre distance au nord ou au sud par rapport à l'équateur, les valeurs du nord étant positives et celles du sud étant négatives. La plage va de -90 à 90. La figure 23-1 montre une carte Google d'un endroit près de Quito, en Équateur. La latitude indiquée sur la carte est -0,01, à peine au sud de l'équateur !



Figure 23-2. Quito, en Équateur, est sur l'équateur

La longitude correspond à quelle distance vous vous trouvez à l'est ou à l'ouest du premier méridien ; les coordonnées est ont des valeurs positives et les coordonnées ouest sont négatives. Le premier méridien, qui s'étend du nord au sud, est situé à Greenwich, une ville près de Londres qui abrite le Royal Observatory, l'organisation qui a initialement établi le premier méridien comme base de mesure pour les astronomes et les navigateurs. La carte de la figure 23-2 montre Greenwich et sa longitude de 0,0.



Figure 23-3. L'Observatoire royal de Greenwich projette un faisceau de lumière le long du premier méridien

Les valeurs de longitude vont de -180 à 180. La figure 23-3 montre un endroit en Russie, très près de l'Alaska, qui a une longitude de 180,0. On pourrait dire qu'un endroit comme celui-ci se trouve à l'autre bout du monde par rapport à Greenwich (0,0 de longitude).

350 Chapitre 23 : Lire et répondre aux capteurs



Figure 23-4. Un point proche de la frontière russo-alaskienne à la longitude 180

DÉTECTER L'EMPLACEMENT AVEC APP INVENTOR

App Inventor fournit le composant LocationSensor pour accéder aux informations GPS.

Le composant possède des propriétés pour la latitude, la longitude et l'altitude. Il communique également avec Google Maps, afin que vous puissiez obtenir une lecture de votre adresse postale actuelle.

LocationSensor.LocationChanged, illustré dans la figure 23-4, est le gestionnaire d'événements clé pour LocationSensor.

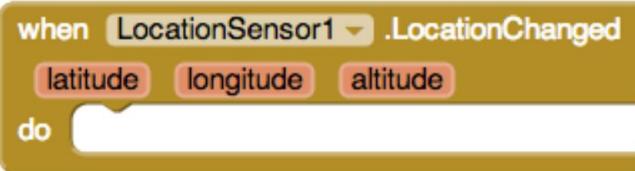


Figure 23-5. Le gestionnaire d'événements LocationSensor1.LocationChanged

Cet événement est déclenché la première fois que le capteur établit une lecture et chaque fois que le téléphone est suffisamment déplacé pour que de nouvelles données soient lues. Il y a souvent un délai de quelques secondes avant la première lecture d'une application, et parfois l'appareil ne peut pas obtenir de lecture du tout. Par exemple, si vous êtes à l'intérieur et que vous n'êtes pas connecté au WiFi, l'appareil risque de ne pas obtenir de lecture. Votre téléphone dispose également de paramètres grâce auxquels vous pouvez activer la lecture GPS pour économiser la batterie ; c'est une autre raison potentielle pour laquelle le composant ne peut pas obtenir de lecture. Pour ces raisons, vous ne devez pas supposer que le LocationSensor

les propriétés ont un paramètre valide jusqu'à l'événement LocationSensor.LocationChanged se produit.

Une façon de gérer les inconnues de la détection de localisation consiste à créer une variable lastKnownLocation, initialisez-le sur « inconnu », puis demandez au gestionnaire d'événements LocationSensor.LocationChanged de modifier la valeur de cette variable, comme indiqué dans la figure 23-5.

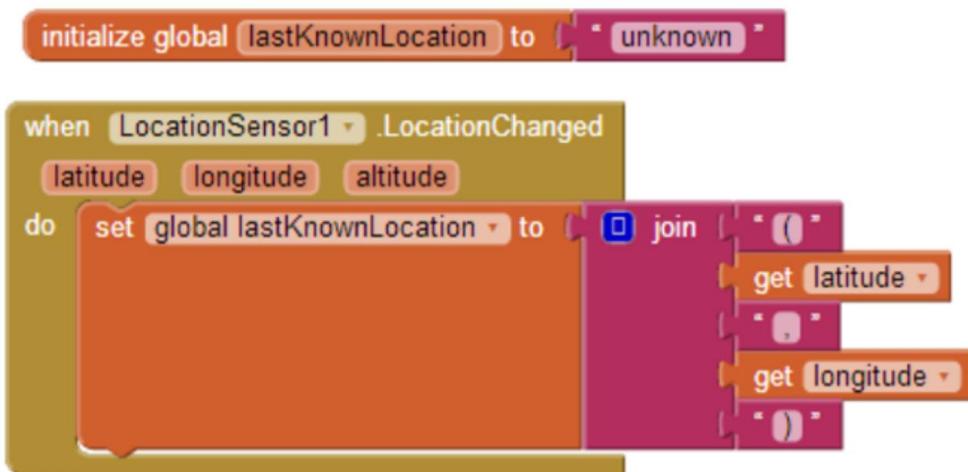


Figure 23-6. La valeur de la variable lastKnownLocation change chaque fois que l'emplacement change

En programmant le gestionnaire d'événements LocationSensor.LocationChanged de cette manière, vous pouvez toujours afficher l'emplacement actuel ou l'enregistrer dans une base de données, avec « inconnu » apparaissant jusqu'à la première lecture. Cette stratégie est utilisée dans le programme No Texting While Driving ! application (Chapitre 4) ; cette application répond automatiquement aux SMS et inclut soit « inconnu » soit la dernière lecture prise dans la réponse.

Vous pouvez également demander explicitement si le capteur a une lecture en utilisant le Bloc LocationSensor.HasLongitudeLatitude illustré dans la figure 23-6.



Figure 23-7. Tester si le capteur a une lecture à l'aide du bloc HasLongitudeLatitude

352 Chapitre 23 : Lire et répondre aux capteurs

VÉRIFICATION DES LIMITES

Une utilisation courante de l' événement LocationChanged consiste à vérifier si l'appareil se trouve dans une limite ou dans une zone définie. Par exemple, considérons le code de la figure 23-7, qui fait vibrer le téléphone chaque fois qu'une nouvelle lecture indique qu'une personne s'est éloignée de plus de 0,1 degré de longitude du premier méridien.

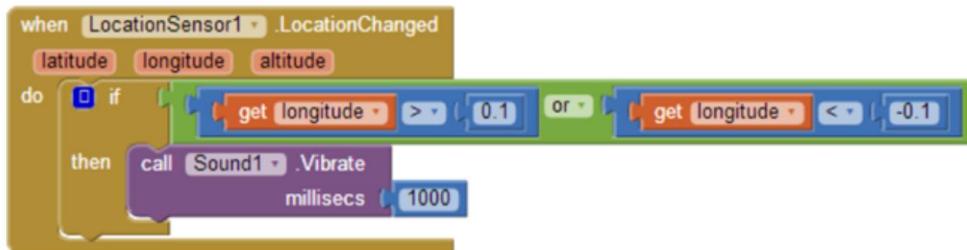


Figure 23-8. Si une lecture n'est pas proche du premier méridien, le téléphone vibre

Une telle vérification des limites a de nombreuses applications ; par exemple, avertir les libérés conditionnels s'ils s'approchent d'une distance légalement spécifiée de leur domicile, ou alerter les parents ou les enseignants si un enfant quitte l'aire de jeux. Si vous souhaitez voir un exemple légèrement plus complexe, consultez la discussion du chapitre 18 sur les blocs conditionnels.

FOURNISSEURS D'INFORMATIONS DE LOCALISATION : GPS, WIFI ET ID CELLULAIRE

Un appareil Android peut déterminer sa propre position de plusieurs manières. La méthode la plus précise, à quelques mètres près, consiste à utiliser les satellites GPS. Cependant, vous n'obtiendrez pas de lecture si vous êtes à l'intérieur ou s'il y a des gratte-ciel ou d'autres obstacles autour de vous ; vous avez besoin d'un chemin clair vers au moins trois satellites du système.

Si le GPS n'est pas disponible ou si l'utilisateur l'a désactivé, l'appareil peut obtenir sa position via un réseau sans fil. Vous devez bien sûr être à proximité d'un routeur WiFi, et la position que vous obtiendrez est la latitude/longitude de cette station WiFi.

Un troisième moyen par lequel un appareil peut déterminer le positionnement consiste à utiliser l'ID de cellule. L'ID de cellule fournit un emplacement du téléphone en fonction de la force des signaux des tours de téléphonie cellulaire à proximité. Ce n'est généralement pas très précis, sauf si vous avez de nombreuses tours de téléphonie cellulaire à proximité de chez vous. Cependant, il utilise le moins d'énergie de la batterie par rapport à la connectivité GPS ou WiFi.

Utilisation du capteur d'orientation

Vous pouvez utiliser l' OrientationSensor pour des applications de type jeu dans lesquelles l'utilisateur contrôle l'action en inclinant l'appareil. Il peut également être utilisé comme boussole pour savoir dans quelle direction (nord/sud, est/ouest) pointe le téléphone.

L' OrientationSensor possède cinq propriétés, qui sont toutes peu familières à la plupart des gens autres que les ingénieurs aéronautiques :

Rouler (gauche-droite)

Le roulis est de 0 degré lorsque l'appareil est de niveau, augmente jusqu'à 90 degrés lorsque l'appareil est incliné vers son côté gauche et diminue jusqu'à -90 degrés lorsque l'appareil est incliné vers son côté droit.

Pas (haut-arrière)

L'inclinaison est de 0 degré lorsque l'appareil est de niveau, augmente jusqu'à 90 degrés lorsque l'appareil est incliné de sorte que son sommet pointe vers le bas, et augmente encore jusqu'à 180 degrés lorsqu'il est retourné. De même, lorsque l'appareil est incliné de manière à ce que son bas pointe vers le bas, le pitch diminue à -90 degrés, puis à -180 degrés lorsqu'il est tourné à fond sur.

Azimut (Boussole)

L'azimut est de 0 degré lorsque le haut de l'appareil pointe vers le nord, de 90 degrés lorsqu'il pointe vers l'est, de 180 degrés lorsqu'il pointe vers le sud et de 270 degrés lorsqu'il pointe vers l'ouest.

Magnitude (Vitesse d'une balle qui roule)

Magnitude renvoie un nombre compris entre 0 et 1 qui indique l'inclinaison de l'appareil. Sa valeur indique la force exercée par une bille roulant sur la surface de l'appareil.

Angle (Angle d'une balle qui roule)

Angle renvoie la direction dans laquelle l'appareil est carrelé. C'est-à-dire qu'il indique la direction de la force qui serait exercée par une bille roulant sur la surface de l'appareil.

L' OrientationSensor fournit l' événement OrientationChanged , qui est déclenché à chaque fois que l'orientation change. Pour explorer ces propriétés plus en détail, écrivons une application qui illustre comment les propriétés changent lorsque l'utilisateur incline l'appareil. Ajoutez simplement cinq étiquettes de titre et cinq autres étiquettes pour afficher les valeurs actuelles des propriétés dans la liste précédente. Ensuite, ajoutez les blocs illustrés à la figure 23-8.

354 Chapitre 23 : Lire et répondre aux capteurs

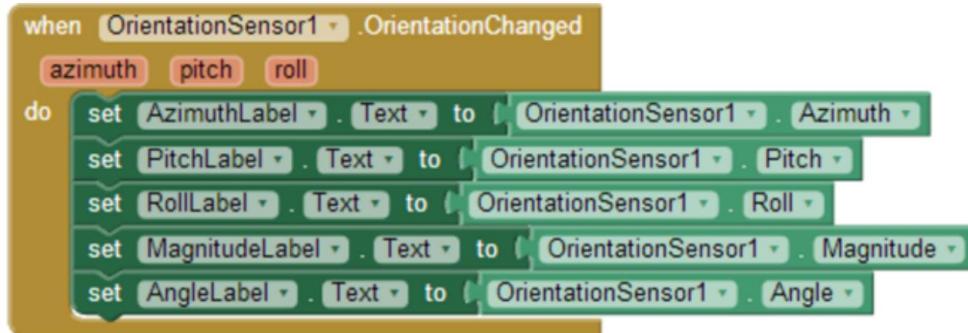


Figure 23-9. Blocs pour afficher les données OrientationSensor

UTILISER LE PARAMÈTRE ROLL POUR DÉPLACER UN OBJET

Cette fois, essayons de déplacer une image vers la gauche ou la droite sur l'écran en fonction de

l'inclinaison de l'appareil par l'utilisateur, comme vous pourriez le faire dans un jeu de tir ou de conduite. Faites glisser un canevas et définissez la largeur sur « Remplir le parent » et la hauteur sur 200 pixels.

Ensuite, ajoutez un ImageSprite ou une Ball dans le canevas, et ajoutez une étiquette nommée RollLabel en dessous pour afficher une valeur de propriété, comme le montre la figure 23-9.



Figure 23-10. Une interface utilisateur pour explorer comment utiliser le rouleau pour déplacer une image

La propriété Roll d' OrientationSensor indiquera si le téléphone est incliné à gauche ou à droite. Si vous tenez le téléphone droit et l'inclinez légèrement vers la gauche, vous obtiendrez une lecture positive du roulis ; si vous l'inclinez légèrement vers la droite, vous obtiendrez une lecture négative. Par conséquent, vous pouvez laisser l'utilisateur déplacer un objet avec un gestionnaire d'événements tel que celui illustré dans la figure 23-10.

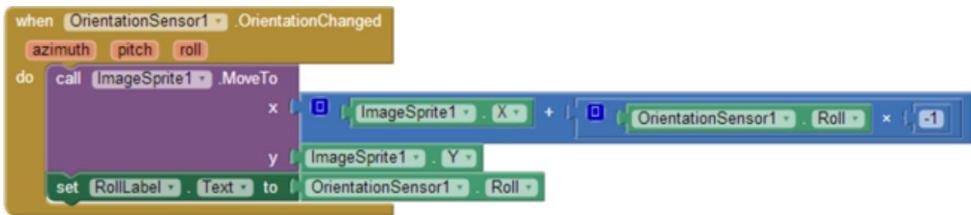


Figure 23-11. Répondre aux modifications de la propriété Roll avec l'événement OrientationChanged

Les blocs multiplient le roulement par -1 , car l'inclinaison vers la gauche donne un roulement positif et devrait déplacer l'objet vers la gauche (rendant ainsi la coordonnée x plus petite). Pour un aperçu du fonctionnement du système de coordonnées dans les applications animées, voir le chapitre 17.

Notez que cette application ne fonctionne que lorsque l'appareil est en mode Portrait (debout), et non en mode Paysage. En l'état, si vous inclinez trop le téléphone, l'écran passera en mode Paysage et l'image restera bloquée sur le côté gauche de l'écran. La raison en est que si l'appareil est sur le côté, il est incliné vers la gauche et obtiendra donc toujours une lecture positive du roulis. Une lecture de rouleau positive, comme indiqué dans les blocs de la figure 23-10, réduira toujours la coordonnée x .

Notez qu'App Inventor fournit la propriété Screen.ScreenOrientation , que vous pouvez utiliser pour verrouiller l'orientation si vous ne souhaitez pas qu'elle bascule entre les modes.

SE DÉPLACER DANS N'IMPORTE QUELLE DIRECTION EN UTILISANT LE CAP ET LA MAGNITUDE

L'exemple de la section précédente déplace l'image vers la gauche ou la droite. Si vous souhaitez autoriser le mouvement dans n'importe quelle direction, vous pouvez utiliser les propriétés Angle et Magnitude de OrientationSensor. Ce sont les propriétés utilisées pour déplacer la coccinelle dans le jeu décrit au chapitre 5.

Dans la figure 23-12, vous pouvez voir les blocs d'une application de test dans laquelle l'utilisateur incline l'appareil pour déplacer un personnage dans n'importe quelle direction (vous avez besoin de deux étiquettes et d'un sprite d'image pour cet exemple).

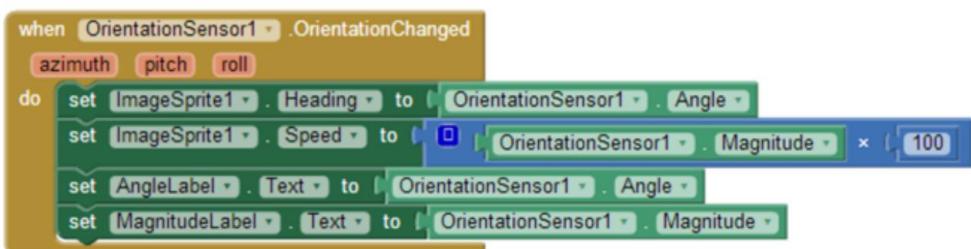


Figure 23-12. Déplacer un personnage en utilisant l'angle et la magnitude

356 Chapitre 23 : Lire et répondre aux capteurs

Essayez celui-ci. La propriété **Magnitude**, une valeur comprise entre 0 et 1, indique comment l'appareil est fortement incliné. Dans cette application de test, l'image se déplace plus rapidement à mesure que la valeur de la magnitude augmente.

UTILISER LE TÉLÉPHONE COMME BOUSSOLE

Les applications Compass et les applications telles que Google Sky Map doivent connaître l'orientation du téléphone dans le monde, est/ouest et nord/sud. Sky Map utilise les informations pour superposer des informations sur les constellations vers lesquelles pointe le téléphone.

La lecture Azimut est utile pour ce type d'orientation. L'azimut est toujours compris entre 0 et 360 degrés, 0 étant le nord ; 90, est ; 180, sud ; et 270, ouest.

Ainsi, une lecture de 45 signifie que le téléphone pointe vers le nord-est, 135 signifie sud-est, 225 signifie sud-ouest et 315 signifie nord-ouest.

Les blocs de la figure 23-12 sont destinés à une simple boussole qui affiche en texte ce qui la direction vers laquelle pointe le téléphone (par exemple, nord-ouest).

Comme vous l'avez peut-être remarqué, les blocs n'affichent qu'une seule des quatre possibilités : nord-ouest, nord-est, sud-ouest et sud-est. Comme défi, voyez si vous pouvez le modifier pour afficher une seule direction (nord, sud, est ou ouest) si la lecture précise que vous pointez à quelques degrés près.

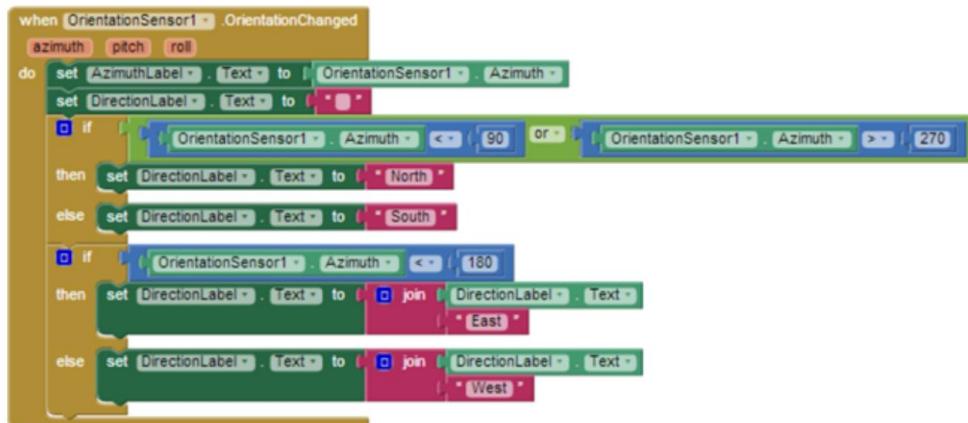


Figure 23-13. Programmation d'une boussole simple

Utiliser l'accéléromètre

L'accélération est le taux de changement de vitesse au fil du temps. Si vous appuyez votre pied sur la pédale d'accélérateur de votre voiture, la voiture accélère : sa vitesse augmente à un rythme particulier.

Un accéléromètre comme celui de votre appareil Android mesure l'accélération, mais son référentiel n'est pas l'appareil au repos, mais plutôt l'appareil en chute libre : si vous

laissez tomber le téléphone, il enregistrera une lecture d'accélération de 0. En termes simples, les lectures prennent en compte la gravité.

Si vous souhaitez en savoir plus sur la physique de la matière, vous devrez consulter vos livres sur Einstein. Mais dans cette section, nous explorerons suffisamment l'accéléromètre pour vous aider à démarrer. Nous examinerons même une application qui pourrait aider à sauver des vies !

RÉPONDRE AUX SECOUATIONS DE L'APPAREIL

Si vous avez terminé l'application Hello Purr au chapitre 1, vous avez déjà utilisé l'AccelerometerSensor. Dans cette application, vous avez utilisé l'événement Accelerometer.Shaking pour faire miauler le chat lorsque le téléphone était secoué, comme le montre la figure 23-13.



Figure 23-14. Jouer un son lorsque le téléphone est secoué

UTILISER LES LECTURES DU CAPTEUR ACCÉLÉROMÈTRE

Comme les autres capteurs, l'accéléromètre a un événement lorsque les lectures changent, AccelerometerSensor.AccelerationChanged. Cet événement a trois arguments correspondant à l'accélération en trois dimensions :

xAccel

Positif lorsque l'appareil est incliné vers la droite (c'est-à-dire que son côté gauche est relevé) et négatif lorsque l'appareil est incliné vers la gauche (son côté droit est relevé).

yAccel

Positif lorsque le bas de l'appareil est relevé et négatif lorsque son haut est relevé.

zAccel

Positif lorsque l'écran de l'appareil est orienté vers le haut et négatif lorsque l'écran est orienté vers le bas.

DÉTECTION DE CHUTE LIBRE

Nous savons que si toutes les lectures d'accélération sont proches de 0, l'appareil est en chute libre vers le sol. Dans cet esprit, nous pouvons détecter un événement de chute libre en vérifiant les lectures de l'événement AccelerometerSensor.AccelerationChanged . Vous pourriez utiliser de tels blocs, avec de nombreux tests, pour détecter la chute d'une personne âgée et envoyer automatiquement un message SMS en réponse.

358 Chapitre 23 : Lire et répondre aux capteurs

La figure 23-14 montre les blocs d'une application qui signale simplement qu'une chute libre a eu lieu. s'est produit (et permet à l'utilisateur de cliquer sur un bouton Réinitialiser pour vérifier à nouveau).¹

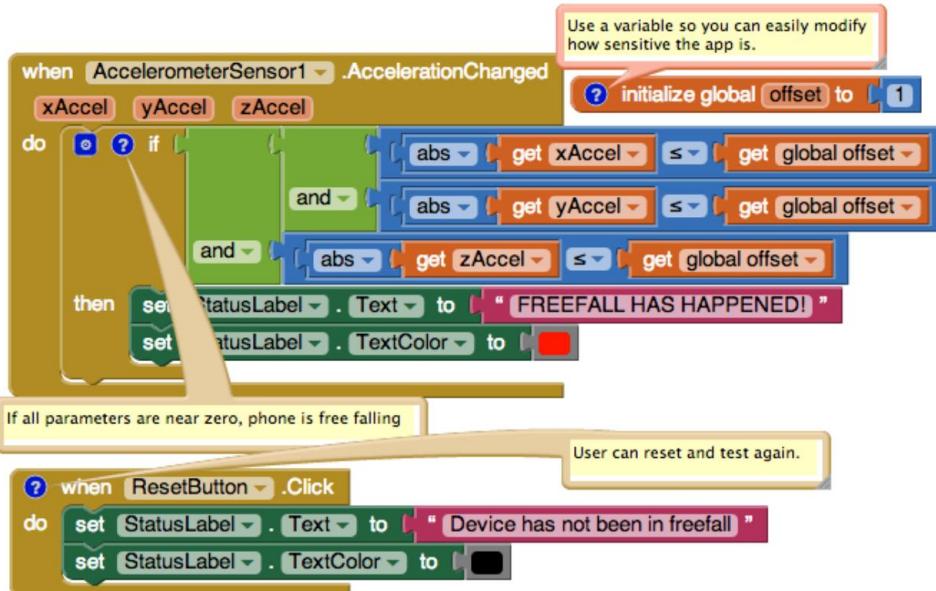


Figure 23-15. Signaler lorsqu'une chute libre s'est produite

Chaque fois que le capteur obtient une lecture, les blocs vérifient les dimensions x, y et z pour voyez s'ils sont proches de 0 (si leur valeur absolue est inférieure à 1). Si les trois sont proches de 0, l'application modifie une étiquette d'état pour indiquer que le téléphone est en chute libre. Lorsque l'utilisateur appuie sur le ResetButton, l'étiquette d'état est réinitialisée à son état d'origine (« L'appareil n'a PAS été en chute libre »).

Résumé

Les capteurs présentent un grand intérêt dans les applications mobiles car ils permettent à vos utilisateurs de véritablement interagir avec leur environnement. En adoptant l'informatique mobile, vous ouvrez tout un monde d'opportunités en matière d'expériences utilisateur et de développement d'applications. Cependant, vous devrez bien réfléchir à comment, où et quand vous utilisez des capteurs dans vos applications. De nombreuses personnes ont des problèmes de confidentialité et pourraient ne pas utiliser votre application si elles s'inquiètent de ce que vous faites avec les données de leurs capteurs. Toujours,

¹ Vous pouvez cliquer avec le bouton droit sur un bloc et choisir « Entrées en ligne » pour modifier la façon dont les blocs apparaissent. Cela a été fait pour les blocs de cet exemple afin de réduire la largeur du gestionnaire d'événements.

avec toutes les options en matière de jeux, de réseaux sociaux, de voyages et bien plus encore, les possibilités de mises en œuvre positives sont presque infinies.

Communiquer avec le Web

La technologie mobile et l'omniprésence du Web ont changé le monde dans lequel nous vivons. Vous pouvez désormais vous asseoir dans le parc et effectuer vos opérations bancaires, rechercher sur Amazon.com pour trouver des critiques du livre que vous lisez et consulter Twitter pour voir quoi auxquels pensent les gens de tous les autres parcs du monde.

Les téléphones mobiles ont bien dépassé le stade des simples appels et SMS : désormais, vous avez également un accès instantané aux données du monde entier.

Vous pouvez utiliser le navigateur de votre téléphone pour accéder au Web, mais le petit écran et la vitesse limitée d'un appareil mobile peuvent souvent rendre cela problématique.

Les applications personnalisées, spécialement conçues pour extraire de petits morceaux d'informations particulièrement adaptées du Web, peuvent constituer une alternative plus attrayante au navigateur mobile.

Dans ce chapitre, nous examinerons les composants App Inventor qui accèdent aux informations à partir du Web. Vous apprendrez comment afficher une page Web dans l'interface utilisateur de votre application, ainsi que les API et comment accéder aux informations à partir d'un service Web.

La créativité consiste à remixer le monde, en combinant (en mélangeant) des idées et du contenu existants de manière nouvelle et intéressante. Eminem fait partie des nombreux artistes au cours des dernières décennies qui ont popularisé le mashup musical en plaçant sa voix Slim Shady sur des morceaux d'AC/DC et de Vanilla Ice. Ce type de « sampling » est désormais courant et de nombreux artistes, dont Girl Talk et Negativland, se concentrent principalement sur la création de nouveaux morceaux en mélangeant d'anciens contenus.

Le monde du Web et celui du mobile ne sont pas différents : les sites Web et les applications remixent le contenu de diverses sources de données, et la plupart des sites sont désormais conçus dans un souci d'interopérabilité. Un exemple illustratif de mashup Web est Housing Maps, illustré dans la figure 24-1, qui prend les informations sur la location d'appartements de Craigslist et les mélange avec l'API Google Maps.

Figure 24-1.



362 Chapitre 24 : Communiquer avec le Web

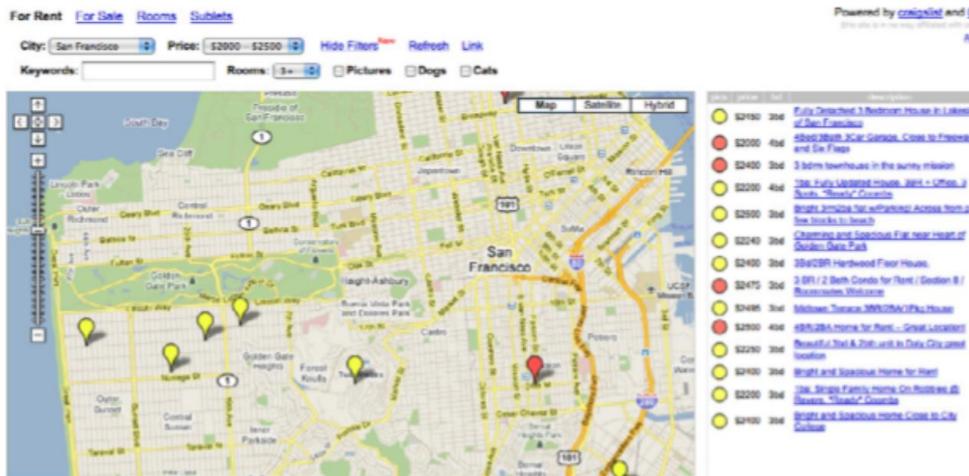


Figure 24-2. Housing Maps rassemble les informations de Craigslist et de Google Maps

Les mashups tels que Housing Maps sont possibles car des services tels que Google Maps fournir à la fois un site Web et une API de service Web correspondante. Nous, les humains, visitons <http://maps.google.com/> dans un navigateur, mais des applications telles que Housing Maps communiquent de machine à machine grâce à l'API Google Maps . Les mashups traitent les données, les combinent avec des données provenant d'autres sites (par exemple, Craigslist), puis les présentent sous une forme nouvelle et intéressante. façons.

Presque tous les sites Web populaires proposent désormais cet accès alternatif de machine à machine. Le programme fournissant les données est appelé service Web, et le protocole permettant à une application cliente de communiquer avec le service est appelé interface de programmeur d'application, ou API. Dans la pratique, le terme API est également utilisé pour désigner le service Web.

L'Amazon Web Service (AWS) a été l'un des premiers services Web, car Amazon s'est rendu compte que l'ouverture de ses données pour une utilisation par des entités tierces entraînerait à terme la vente de davantage de livres. Lorsque Facebook a lancé son API en 2007, de nombreuses personnes ont haussé les sourcils. Les données de Facebook ne sont pas des publicités pour des livres, alors pourquoi devrait-il laisser d'autres applications « voler » ces données et potentiellement détourner de nombreux utilisateurs du site Facebook (et de ses publicités !) ? Pourtant, son ouverture a conduit Facebook à devenir une plate-forme plutôt qu'un simple site, ce qui signifie que d'autres programmes pourraient s'appuyer sur les fonctionnalités de Facebook et en exploiter les fonctionnalités, et personne ne peut contester son succès aujourd'hui. Au moment du lancement de Twitter en 2009, l'accès aux API était une attente, pas une nouveauté, et Twitter a agi en conséquence. Désormais, comme le montre la figure 24-2, la plupart des sites Web proposent à la fois une API et une interface humaine.

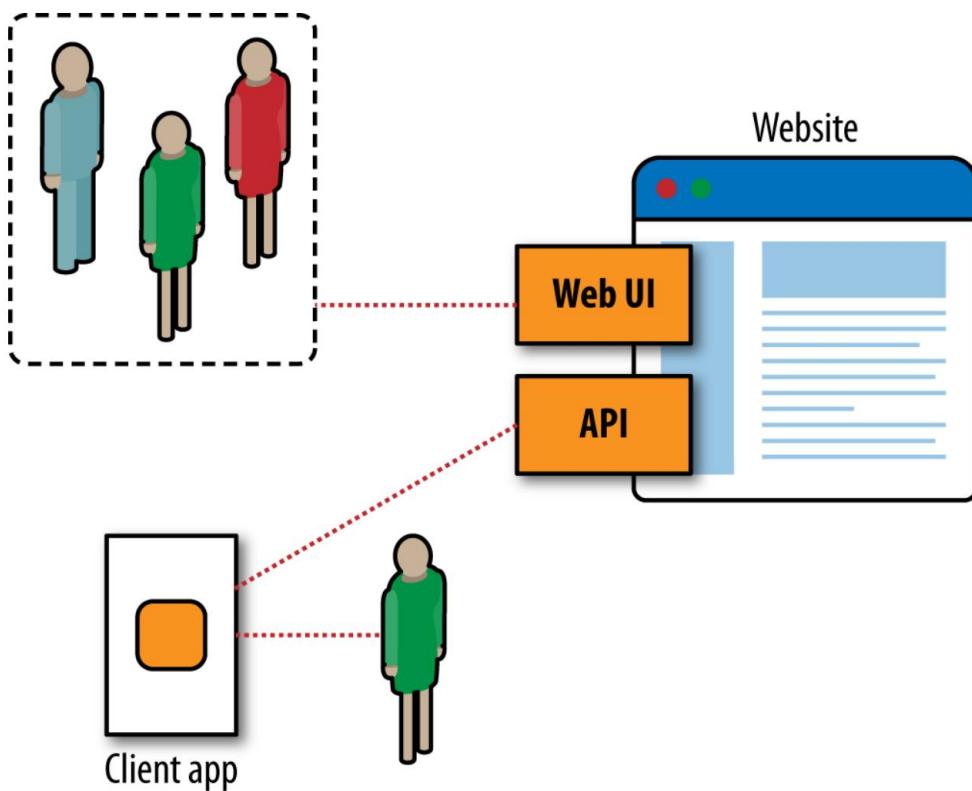


Figure 24-3. La plupart des sites Web fournissent à la fois une interface humaine et une API pour les applications clientes

Ainsi, le Web est une chose pour nous, les humains moyens (un ensemble de sites à visiter). À programmeurs, il s'agit de la base de données d'informations la plus vaste et la plus diversifiée au monde.

Le composant WebViewer

Le composant WebViewer vous permet d'afficher une page Web dans votre application. Vous pouvez afficher une page Google Maps indiquant l'emplacement actuel de l'utilisateur, une page Twitter affichant les sujets de tendance les plus récents liés à votre application ou une page de nba.com affichant les statistiques de vos joueurs préférés.

WebViewer (voir Figure 24-3) est comme le composant Canvas dans le sens où il définit un sous-panneau de l'écran. Mais alors que Canvas est utilisé pour les dessins et les animations, WebViewer affiche une page Web.

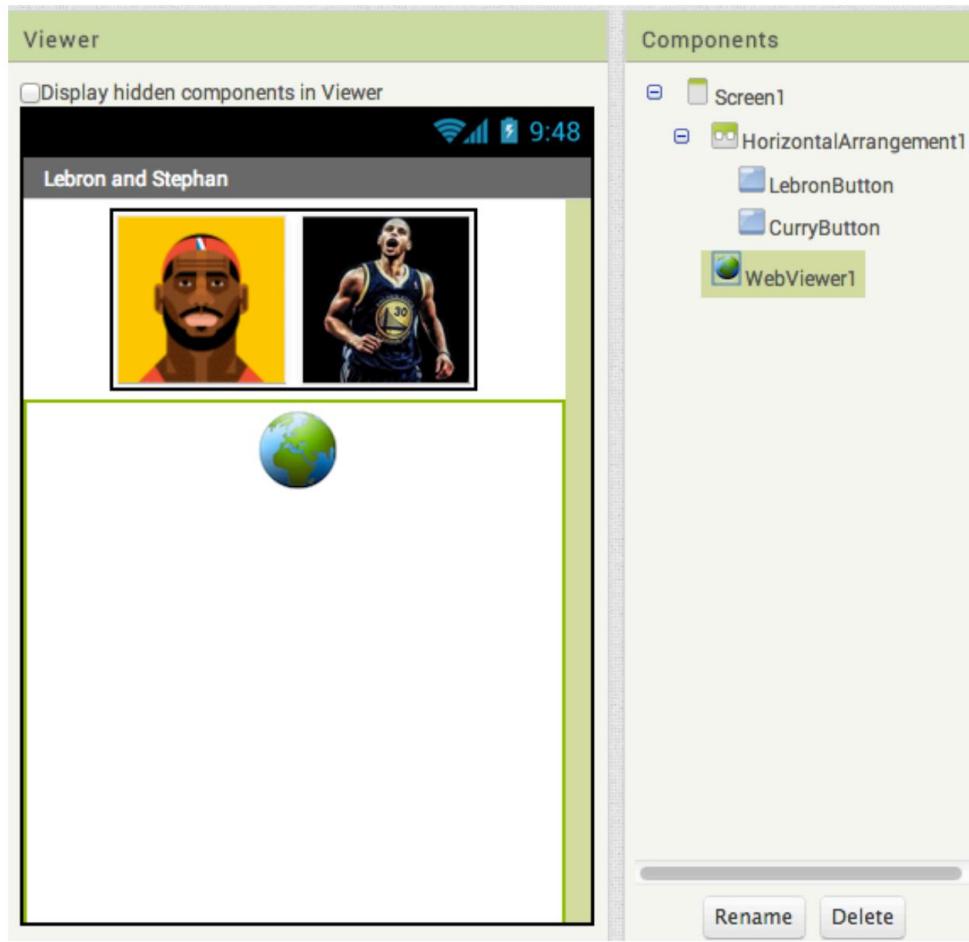


Figure 24-4. Le WebViewer tel qu'il apparaît dans Designer.

Vous pouvez faire glisser un WebViewer depuis le tiroir de l'interface utilisateur. Vous pouvez ensuite modifier dynamiquement l'URL qui apparaît, comme dans la figure 24-4, qui représente les blocs d'une application affichant les statistiques des joueurs NBA LeBron James et Stephen Curry :

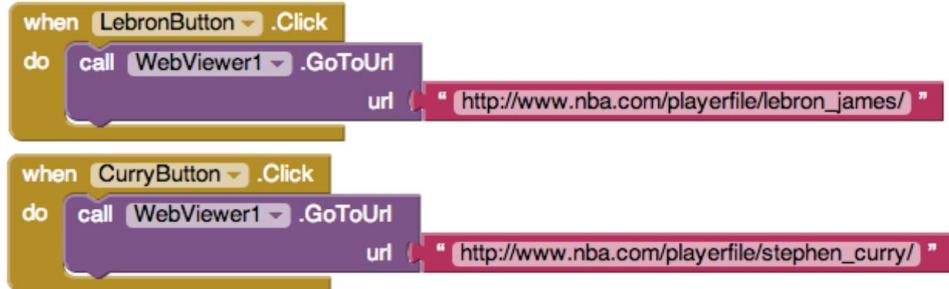


Figure 24-5. Blocs pour afficher la page Web des joueurs choisis

Si l'utilisateur appuie sur la photo de Stephen Curry, l'application affichera sa page de nba.com dans WebViewer, comme dans la figure 24-5.

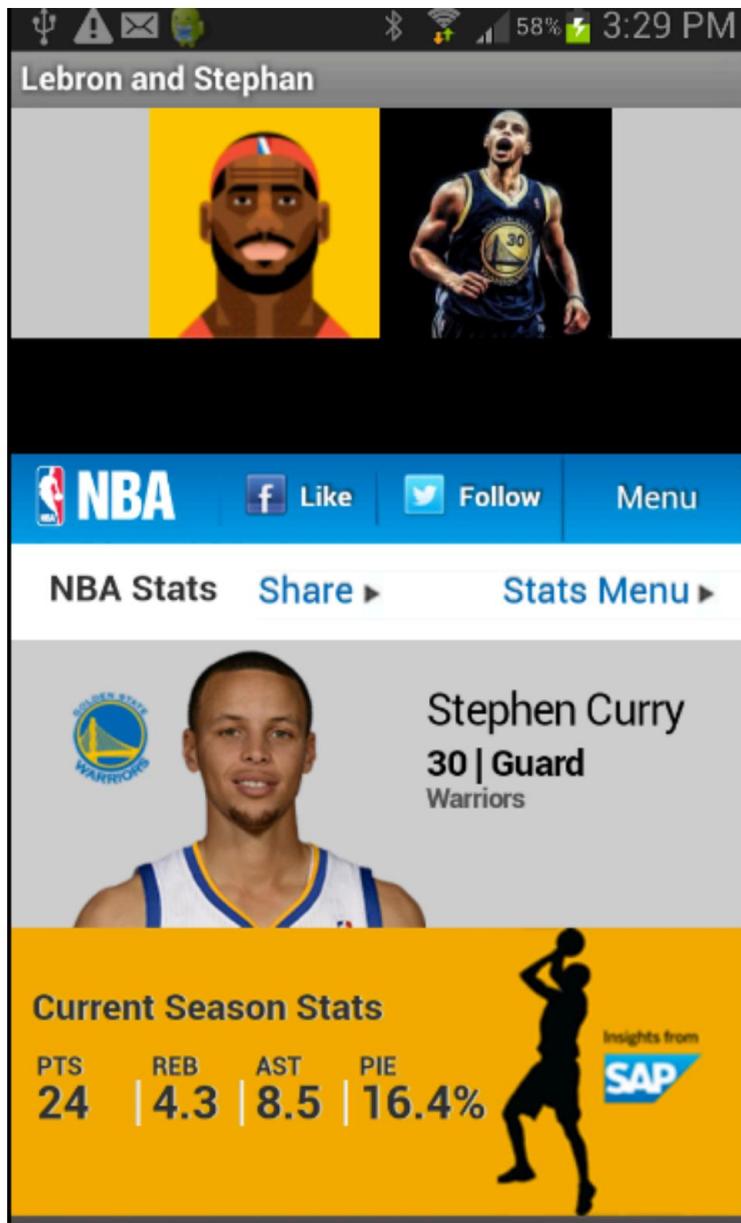


Figure 24-6. WebViewer dans l'application

Le composant Web

Alors que WebViewer affiche une page Web, le composant Web , un composant relativement nouveau dans App Inventor, facilite la communication d'une application avec un service Web via le protocole HTTP (Hypertext Transfer Protocol) standard. Ce protocole fournit Get, Put et

Publiez des méthodes pour importer des informations dans votre application. Les informations n'arrivent pas sous forme de page affichable, mais sous forme de données que vous pouvez afficher ou traiter à votre guise.

Le composant est de niveau assez bas et son utilisation nécessite une certaine expertise en programmation. Vous définissez généralement la propriété Web.URL pour spécifier le service Web avec lequel vous communiquerez, puis vous appelez l'une des méthodes HTTP pour demander une action. C'est compliqué car vous devez comprendre l'API du service Web (le protocole de communication) et vous devez comprendre comment traiter les informations que le service Web renvoie à votre application. Ce traitement est connu sous le nom d'analyse syntaxique et il s'agit d'une technique de programmation avancée.

Dans ce chapitre, vous découvrirez le composant Web à travers un exemple relativement simple qui accède aux informations sur le cours des actions financières à partir d'une API publique mise à disposition par Yahoo Finance. Le protocole pour communiquer avec cette API est assez simple et les données renvoyées se trouvent dans une liste de valeurs séparées par des virgules (valeurs séparées par des virgules ou CSV), ce qui constitue donc une belle introduction à la communication API.

Malheureusement, la plupart des API ont des schémas d'autorisation et des API complexes, et elles renvoient souvent des données dans des formats tels que JavaScript Object Notation (JSON) ou XML, qui nécessitent un code avancé pour être analysés.

ÉCHANTILLON DE BOURSE

La figure 24-6 montre les blocs d'une application qui affiche les informations boursières de Google au lancement de l'application.

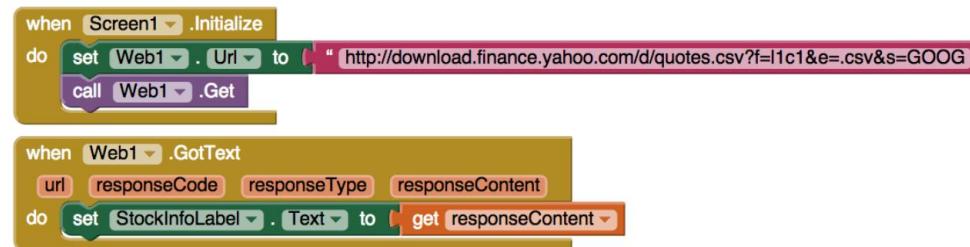


Figure 24-7. Accéder aux informations sur le stock en direct via le composant Web

Sur Screen.Initialize, Web1.Url est défini sur l'URL permettant de communiquer avec Yahoo Finance. Lorsque Web1.Get est appelé, la demande est effectuée, mais aucune donnée n'est renvoyée immédiatement.

Au lieu de cela, lorsque Yahoo renvoie les données demandées à votre application, l'événement Web1.GotText est déclenché et c'est ici que vous pouvez traiter les données renvoyées. Le paramètre d'événement ResponseContent contient les données. Comme nous venons de le mentionner, l'API Yahoo Finance renvoie les données au format CSV. Si vous créez cette application et l'exécutez, vous verrez que la version actuelle

Le cours de l'action Google et l'évolution du cours du jour sont affichés dans StockInfoLabel, séparés par des virgules.

Vous pouvez personnaliser le Web.Url pour obtenir les informations sur une ou plusieurs sociétés différentes et pour obtenir différents types d'informations boursières. L'API Yahoo Finance, sur <https://code.google.com/p/yahoo-finance-managed/wiki/CSVAPI>, précise comment vous pouvez modifier l'URL pour personnaliser votre demande, ainsi que le format des données qu'elle contient.

Retour.

API TinyWebDB et TinyWebDB compatibles

Le composant Web fournit une méthode d'accès aux API. Si une API est assez simple, comme Yahoo Finance, les programmeurs débutants peuvent utiliser le composant Web pour y accéder directement. Mais d'autres API, comme l'API Amazon présentée au chapitre 13, sont plus compliquées.

Pour les API complexes, un programmeur expérimenté peut configurer un service Web compatible TinyWebDB qui peut ensuite être utilisé par des programmeurs App Inventor moins expérimentés pour accéder à l'API. Lorsqu'un tel service est configuré, d'autres programmeurs peuvent accéder au service Web avec le simple protocole de valeur de balise inhérent à la fonction TinyWebDB.GetValue . Vous envoyez une balise particulière en tant que paramètre et une liste ou un objet texte est renvoyé en tant que valeur. De cette façon, le programmeur App Inventor est protégé de la programmation difficile requise pour analyser (comprendre et extraire des données) les formats de données standard tels que XML ou JSON.

« Conforme à TinyWebDB » signifie simplement un service Web qui suit le protocole attendu de TinyWebDB : il attend une requête spécifique et renvoie des données que TinyWebDB peut comprendre. Le service Web API Amazon utilisé au chapitre 13 est un exemple d'un tel service Web et peut être utilisé comme exemple pour les programmeurs qui souhaitent configurer un tel service (par exemple, si vous êtes enseignant et souhaitez fournir un accès à une API pour vos étudiants).

Dans le passé, créer des API était difficile car vous deviez non seulement comprendre la programmation et les protocoles Web, mais vous deviez également configurer un serveur pour héberger votre service Web et une base de données pour stocker les données. Désormais, c'est beaucoup plus simple car vous pouvez exploiter des outils de cloud computing tels que App Engine de Google et Elastic Compute Cloud d'Amazon pour déployer immédiatement le service que vous créez. Ces plates-formes hébergeront non seulement votre service Web, mais permettront également à des centaines d'utilisateurs d'y accéder avant de vous facturer un seul centime. Comme vous pouvez l'imaginer, ces sites sont une véritable aubaine pour l'innovation.

Les détails de la création d'un service Web compatible TinyWebDB dépassent le cadre de ce livre. Mais si vous êtes intéressé, consultez la documentation et les exemples sur <http://appinventorapi.com/>.

Résumé

La plupart des sites Web et de nombreuses applications mobiles ne sont pas des entités autonomes ; pour faire leur travail, ils s'appuient sur l'interopérabilité des autres sites. Avec App Inventor, vous pouvez créer des jeux, des quiz et d'autres applications autonomes, mais bientôt, vous rencontrerez des problèmes liés à l'accès au Web. Puis-je écrire une application qui m'indique quand le prochain bus arrivera à mon arrêt habituel ? Puis-je écrire une application qui envoie des SMS à un sous-ensemble spécial de mes amis Facebook ? Puis-je écrire une application qui envoie des tweets ? App Inventor fournit trois composants capables de communiquer avec le Web : le WebViewer pour afficher une page Web en direct ; le composant Web , pour accéder aux informations depuis une API ; et le composant TinyWebDB pour accéder aux données dans une API Web spécialement conçue.

Accéder à une API peut être compliqué ; vous devez connaître le protocole de demande d'informations et traiter (analyser) les données souvent complexes renvoyées. Mais la récompense pour avoir appris à faire cela est grande ; vos applications peuvent interagir avec le monde !

