РОЗДІЛ 5

Погоня за сонечком

Малюнок 5-1.

Ігри є одними з найцікавіших додатків для мобільних пристроїв, як для гри, так і для створення. Нещодавній хіт Angry Birds був завантажений 50 мільйонів разів за перший рік свого існування, і в нього грають понад мільйон годин щодня, згідно з даними компанії Rovio, розробника гри. (Існують навіть розмови про те, щоб зробити з нього художній фільм!) Хоча ми не можемо гарантувати такий успіх, ми можемо допомогти вам створювати власні ігри за допомогою App Inventor, включаючи цю, в якій сонечко їсть попелицю, уникаючи жаби.

Що ви побудуєте

У цій "жувальній" грі від першої особи користувачеві належить
представлений сонечком, рух якого контролюватиметься нахилом пристрою. Це вводить користувача в гру інакше, ніж у MoleMash (*Розділ 3), де* користувач перебував поза пристроєм, простягаючи руку всередину.

Додаток Ladybug Chase показано на *Рисунку 5-1*. Користувач може:

- Керуйте сонечком, нахиляючи пристрій.
- На екрані з'являється смужка рівня енергії, яка з часом зменшується, що призводить до голодування сонечка.
- Нехай сонечко переслідує та їсть попелицю, щоб отримати енергію та запобігти голоду.
- Допоможіть сонечку уникнути жаби, яка хоче його з'їсти.

Малюнок 5-2. Гра "Погоня за сонечком" у конструкторі

Що ви дізнаєтесь

Перш ніж заглиблюватися у цю главу, вам слід попрацювати з додатком MoleMash у *Розділі 3,* оскільки він передбачає, що ви знаєте про створення процедур, генерацію випадкових чисел, блок `if-then-else`, а також компоненти `ImageSprite`, `Canvas`, `Sound` і `Clock`.

На додаток до огляду матеріалу з MoleMash та інших попередніх розділів, ця глава вводить в курс справи:

- Використання декількох компонентів `ImageSprite` та виявлення колізій між ними.
- Виявлення нахилів пристрою за допомогою компонента `OrientationSensor` і використання його для керування `ImageSprite`.
- Зміна зображення, що відображається для `ImageSprite`.
- Малювання ліній на компоненті `Canvas`.
- Керування кількома подіями за допомогою компонента `Clock`.
- Використання змінних для відстеження чисел (рівень енергії сонечка).
- Створення та використання процедур з параметрами.
- Використовуючи блок `i`.

Проектування компонентів

Цей додаток матиме `Canvas`, що забезпечує ігрове поле для трьох компонентів `ImageSprite`: один для сонечка, один для попелиці та один для жаби, яка також потребуватиме компонента `Sound` для свого "риб'ячого кроку". Датчик `орієнтації` буде використовуватися для вимірювання нахилу пристрою для переміщення сонечка, а `годинник` - для зміни напрямку руху попелиці.

З'явиться друге `полотно`, на якому відображатиметься рівень енергії сонечка. Кнопка Reset перезапустить гру, якщо сонечко зголодніє або його з'їдять. У *Таблиці 5-1* наведено повний перелік компонентів цього додатка.

Таблиця 5-1. Всі компоненти для гри "Погоня за сонечком

Тип компонента	Група палітри	Як ви його назвете	Мета
Полотно	Малювання та Анімація	`FieldCanvas`	Ігрове поле.

Тип компонента	Група палітри	Як ви його назвете	Мета
ImageSprite	Малювання та Анімація	Сонечко.	Гравець, керований користувачем.
Датчик орієнтації	Датчики	OrientationSensor1	Визначайте нахил телефону, щоб керувати сонечко.
Годинник	Інтерфейс користувача	Годинник 1	Визначає, коли потрібно змінити Заголовки ImageSprites.
ImageSprite	Малювання та Анімація	Попелиця	Здобич сонечка.
ImageSprite	Малювання та Анімація	Жаба.	Сонечко - хижак.
Полотно	Малювання та Анімація	EnergyCanvas	Відобразити рівень енергії сонечка.
Кнопка	Інтерфейс користувача	RestartButton	Перезапустіть гру.
Звук	Медіа	Звук 1	"Кролик", коли жаба з'їдає сонечко.

ПОЧАТОК РОБОТИ

Завантажте наступні файли:

- *http://appinventor.org/bookFiles/LadybugChase/ladybug.png*
- *http://appinventor.org/bookFiles/LadybugChase/aphid.png*
- *http://appinventor.org/bookFiles/LadybugChase/dead_ladybug.png*
- *http://appinventor.org/bookFiles/LadybugChase/frog.png*
- *http://appinventor.org/bookFiles/LadybugChase/frog.wav*

Це зображення *сонечка*, *попелиці*, *мертвого сонечка* та *жаби,* а також звуковий файл для жаб*'ячого квакання*. Завантаживши їх на свій комп'ютер, додайте їх до свого додатку в розділі "Медіа" конструктора.

Підключіться до веб-сайту App Inventor і створіть новий проект. Назвіть його "LadybugChase", а також встановіть заголовок екрана на "Ladybug Chase". Відкрийте редактор блоків і підключіться до пристрою.

РОЗМІЩЕННЯ ПОЧАТКОВИХ КОМПОНЕНТІВ

Хоча у попередніх розділах ви створювали всі компоненти одночасно, розробники зазвичай працюють не так. Натомість, більш поширеним є створення однієї частини програми за раз, її тестування, а потім перехід до наступної частини програми. У цьому розділі ми створимо сонечко і будемо керувати його

рухом.

- У Дизайнері створіть полотно, назвіть його FieldCanvas і встановіть його ширину як "Батько заливки", а висоту - 300 пікселів.
- Розмістіть ImageSprite на полотні, перейменувавши його на Ladybug і встановивши його властивість Picture на зображення сонечка. Не турбуйтеся про значення властивостей X і Y, оскільки вони залежать від того, де на полотні ви розмістили ImageSprite.

Як ви могли помітити, ImageSprites також мають інтервал, напрямок і швидкість властивості, які ви будете використовувати в цій програмі:

- Властивість Interval, яку для цієї гри ви можете встановити рівною 10 (мілісекунд), визначає, як часто ImageSprite має рухатися сам (на відміну від переміщення процедурою MoveTo, яку ви використовували для MoleMash).
- Властивість Heading вказує напрямок, у якому ImageSprite має рухатися, у градусах. Наприклад, 0 означає "праворуч", 90 - "прямо", 180 - "ліворуч" і так далі. Не змінюйте значення параметра Heading; ми змінимо його у редакторі блоків.
- Властивість Speed (Швидкість) визначає, на скільки пікселів має переміститися ImageSprite щоразу, коли минає його інтервал (10 мілісекунд). Ми також налаштуємо властивість Speed у редакторі блоків.

Рух сонечка буде контролюватися датчиком орієнтації, який визначає нахил пристрою. Ми хочемо використовувати компонент Clock для перевірки орієнтації пристрою кожні 10 мілісекунд (100 разів на секунду) і відповідно змінювати напрямок руху сонечка. Ми налаштуємо його у редакторі блоків наступним чином:

1. Додайте OrientationSensor, який з'явиться в розділі "Невидимі компоненти".
2. Додайте годинник, який також з'явиться у розділі "Невидимі компоненти", і встановіть для нього параметр TimerInterval на 10 мілісекунд. Перевірте додані вами зміни за допомогою *Рисунок 5-2*.

Якщо ви використовуєте пристрій, відмінний від емулятора, вам потрібно вимкнути автоматичне обертання екрана, яке змінює напрямок відображення при повороті пристрою. Виберіть екран Screen1 і встановіть для його властивості ScreenOrientation значення Portrait (Книжкова орієнтація).

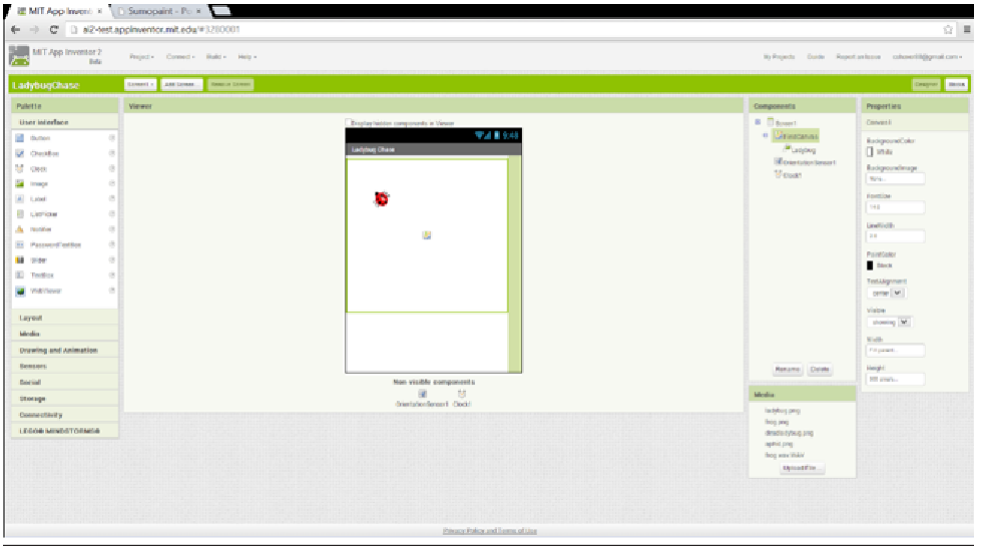


Рисунок 5-3. Налаштування користувацького інтерфейсу у Дизайнері компонентів для анімації сонечка

Додавання поведінки до компонентів

ПЕРЕСУВАЮЧИ СОНЕЧКО.

Перейшовши до редактора блоків, створіть процедуру `UpdateLadybug` і блок `Clock1.Timer`, як показано на *рисунку 5-3*. Спробуйте ввести назви деяких блоків (наприклад, "Clock1.Timer") замість того, щоб перетягувати їх з шухляд. (Зверніть увагу, що до числа 100 застосовується операція множення, позначена зірочкою, яку може бути важко побачити на рисунку). Вам не потрібно створювати жовті підказки до коментарів, хоча ви можете це зробити, клацнувши правою кнопкою миші на блоці і вибравши пункт "Додати коментар".

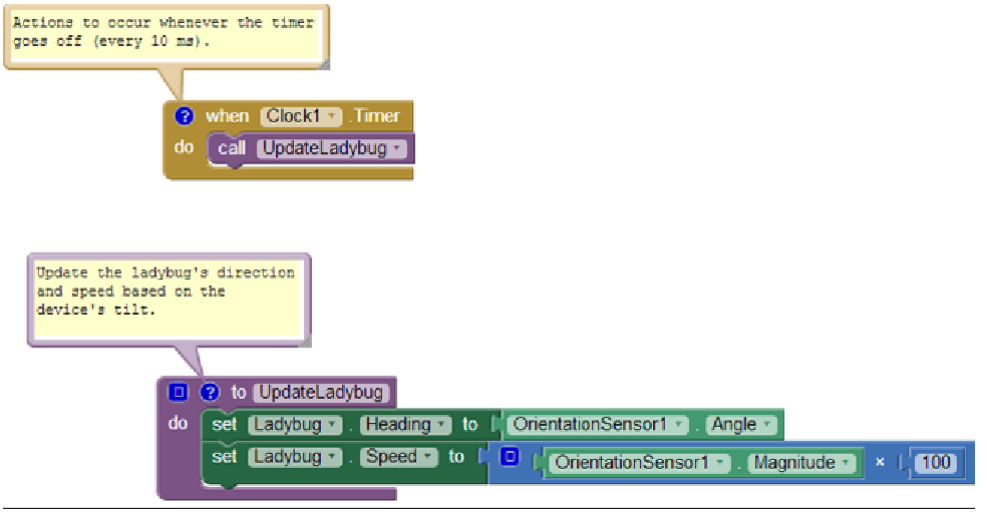
Процедура `UpdateLadybug` використовує дві найбільш корисні властивості `OrientationSensor`:

- `Кут`, який вказує напрямок нахилу пристрою (в градусах).
- Величина, яка вказує на величину нахилу, в діапазоні від 0 (немає нахилу) до 1 (максимальний нахил).

Множення `Magnitude` на 100 вказує сонечку, що він повинен рухатися в межах від 0 до 100 пікселів у вказаному напрямку (`Heading`) щоразу, коли минає інтервал `TimerInterval`, який ви попередньо встановили на 10 мілісекунд у Компонентному Дизайнері.

Хоча ви можете спробувати це на підключеному пристрої, рух сонечка може бути повільнішим і ривками, ніж якщо ви запакуєте і завантажите додаток на пристрій. Якщо після цього ви вважаєте, що сонечко рухається надто повільно,

збільште множник швидкості. Якщо сонечко здається занадто ривками, зменшіть його.



Малюнок 5-4. Зміна напрямку та швидкості руху сонечка кожні 10 мілісекунд

ВІДОБРАЖЕННЯ РІВНЯ ЕНЕРГІЇ

Ми відобразимо рівень енергії сонечка за допомогою червоної смужки на другому полотні. Лінія буде заввишки 1 піксель, а її ширина дорівнюватиме кількості пікселів енергії сонечка, яка коливається від 200 (сите) до 0 (мертве).

Додавання компонента

У Дизайнері створіть нове Полотно. Розмістіть його під `FieldCanvas` і назвіть так `EnergyCanvas`. Встановіть для його властивості `Width` значення "Fill parent", а для властивості `Height` - 1 піксель.

Створення змінної: energy

У редакторі блоків вам потрібно створити змінну `energy` з початковим значенням 200, щоб відстежувати рівень енергії сонечка, як показано на *рисунку 5-4*. (Як ви пам'ятаєте, ми вперше використали змінну `dotSize` у програмі PaintPot у *Главі 2*). Ось як це зробити:

1. У редакторі блоків перетягніть ініціалізовану глобальну назву до блоку. Змініть текст "name" на "energy".
2. Створіть блок з числом `200` (або почавши вводити число 200, або перетягнувши його з математичної панелі) і підключіть його до глобальної енергії, як показано на *рисунку 5-4*.

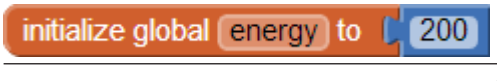
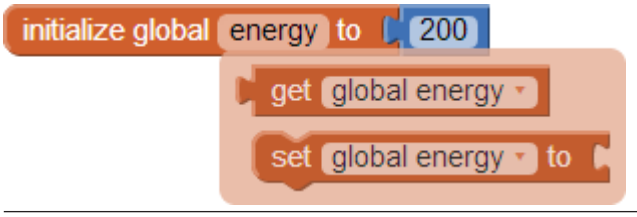


Рисунок 5-5. Ініціалізація змінної енергії на 200

На рисунку 5-5 показано, що коли ви визначаєте змінну, для неї створюються нові блоки `set` і `get`, до яких ви можете отримати доступ, навівши курсор миші на ім'я змінної.



Малюнок 5-6. При наведенні миші на змінну у глобальному блоці ініціалізації, ви можете перетягнути набір і отримати блоки для змінної

Малювання енергетичного батончика

Ми хочемо позначити рівень енергії червоною смужкою, довжина якої в пікселях дорівнює значенню енергії. Для цього ми можемо створити два схожих набори блоків наступним чином:

1. Намалюйте червону лінію від (0, 0) до (`energy`, 0) у `FieldCanvas`, щоб показати поточний рівень енергії.
2. Намалюйте білу лінію від (0, 0) до (`EnergyCanvas.Width`, 0) у `FieldCanvas`, щоб стерти поточний рівень енергії перед малюванням нового рівня.

Однак кращою альтернативою є створення процедури, яка може намалювати лінію будь-якої довжини і будь-якого кольору у `FieldCanvas`. Для цього ми повинні вказати два параметри, довжину і колір, коли викликаємо нашу процедуру, так само, як нам потрібно було вказати значення параметрів у MoleMash, коли ми викликали вбудовану процедуру випадкового цілого числа. Нижче наведено кроки для створення процедури `DrawEnergyLine`:

1. Перейдіть до панелі процедур і перетягніть блок `to procedure do`. Виберіть версію, яка містить "do", а не "return", оскільки наша процедура не повертатиме значення.
2. Клацніть назву процедури ("procedure") і змініть її на "DrawEnergyLine".
3. У верхньому лівому куті нового блоку натисніть на маленький синій квадрат. Відкриється вікно, показане ліворуч на *Мал. 5-6*.
4. З лівого боку цього вікна перетягніть вхідні дані до правого боку, змінивши його назву з "x" на "length". Це означає, що процедура матиме параметр з назвою "length".
5. Повторіть це для другого параметра з назвою "color", який має бути під параметром з назвою "length". Це має виглядати так, як показано праворуч на *Рисунку 5-6*.

6. Натисніть синю іконку ще раз, щоб закрити вікно входів.
7. Заповніть решту процедури, як показано на *малюнку 5-7*. Ви можете знайти отримати колір і отримати довжину можна, навівши курсор миші на їхні імена у визначенні процедури.

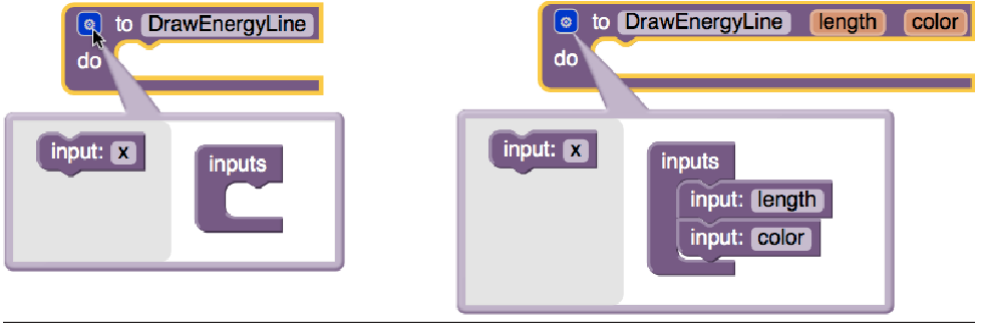


Рисунок 5-7. Додавання входів (параметрів) до процедури DrawEnergyLine

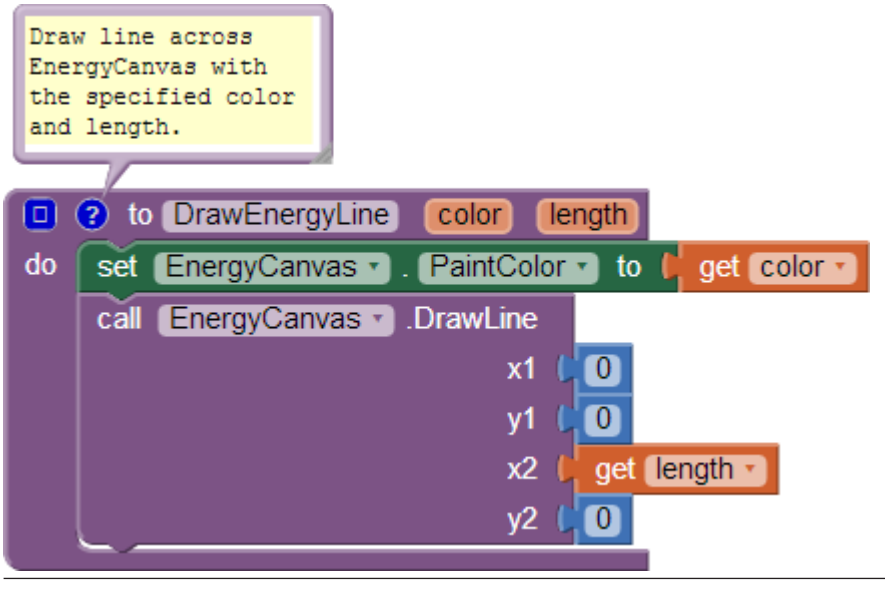


Рисунок 5-8. Визначення процедури DrawEnergyLine

Тепер, коли ви навчилися створювати власні процедури, давайте також напишемо процедуру `DisplayEnergy`, яка викликає `DrawEnergyLine` двічі: один раз, щоб стерти стару лінію (намалювавши білу лінію по всьому полотну), а другий раз, щоб відобразити нову лінію, як показано на *рисунку 5-8*.

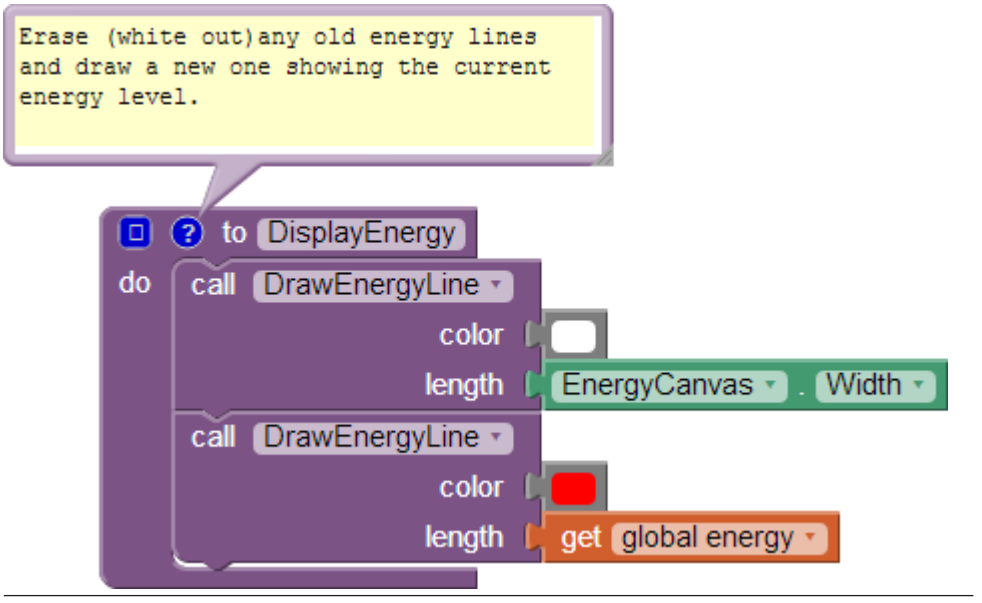


Рисунок 5-9. Визначення процедури DisplayEnergy

Процедура `DisplayEnergy` складається з чотирьох рядків, які виконують наступні дії:

1. Встановіть білий колір фарби.
2. Намалюйте лінію по всій поверхні `EnergyCanvas` (висотою лише 1 піксель).
3. Встановіть червоний колір фарби.
4. Намалюйте лінію, довжина якої в пікселях дорівнює значенню енергії.

Примітка *Процес заміни загального коду викликами нової процедури називається* рефакторингом - *набором потужних технік для того, щоб зробити програми більш зручними для супроводу та надійними. У цьому випадку, якщо ми захочемо змінити висоту або розташування енергетичної лінії, нам достатньо буде внести одну зміну в* `DrawEnergyLine`, *замість того, щоб вносити зміни в кожен її виклик.*

ГОЛОД

На відміну від додатків у попередніх розділах, ця гра може закінчитися: вона закінчиться, якщо сонечко не з'їсть достатньо попелиці або якщо його з'їсть жаба. У будь-якому з цих випадків ми хочемо, щоб сонечко перестало рухатись (це можна зробити, встановивши `Ladybug.Enabled` у false), а картинка змінилася з

живої сонечка на мертву

один (що ми можемо зробити, змінивши Ladybug.Picture на ім'я відповідного завантаженого зображення). Створіть процедуру GameOver, як показано на *рисунку 5-9*.

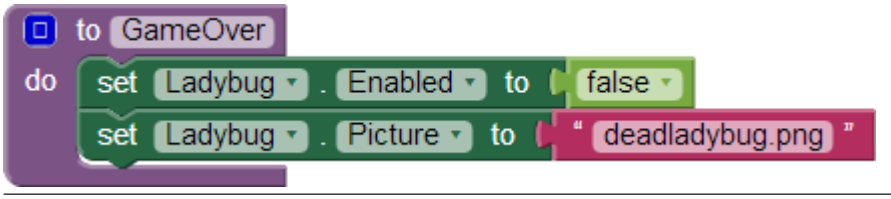


Рисунок 5-10. Визначення процедури GameOver

Далі додайте код, позначений червоним на *рисунку 5-10*, до UpdateLadybug (який, як ви пам'ятаєте, викликається Clock.Timer кожні 10 мілісекунд), як показано нижче:

- Знижуйте його енергетичний рівень.
- Відобразити новий рівень.
- Завершіть гру, якщо енергія дорівнює 0.

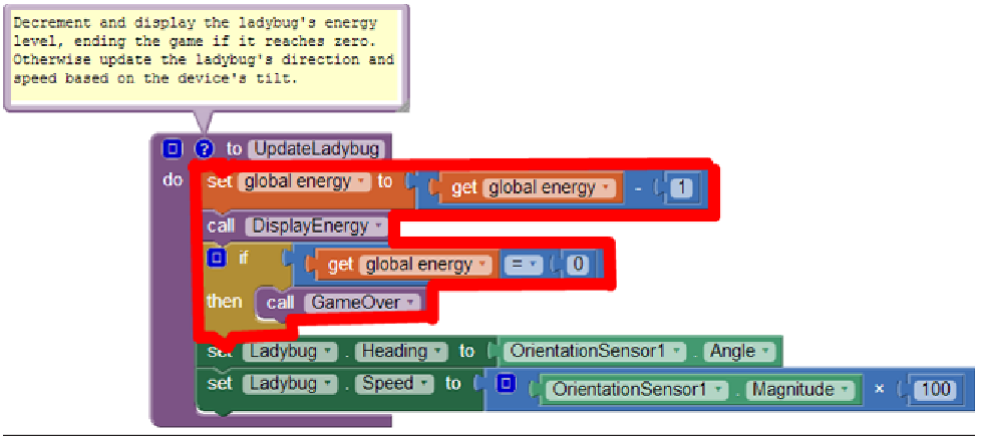


Рисунок 5-11. Друга версія процедури UpdateLadybug

Протестуйте свій додаток *на своєму пристрої і переконайтеся, що рівень енергії з часом зменшується, що врешті-решт призводить до загибелі сонечка.*

ДОДАВАННЯ ПОПЕЛИЦІ

Наступним кроком буде додавання попелиці. Зокрема, попелиця повинна пурхати навколо FieldCanvas. Якщо сонечко зіткнеться з попелицею (таким чином "з'їсть" її), рівень енергії сонечка має підвищитися, а попелиця зникнути, а на її місці трохи

згодом з'явиться інша

пізніше. (З точки зору користувача, це буде інша попелиця, але насправді це буде той самий компонент ImageSprite).

Додавання ImageSprite

Перший крок, який потрібно зробити, щоб додати попелицю, - це повернутися до Дизайнера і створити ще один ImageSprite, намагаючись не розміщувати його поверх сонечка. Його слід перейменувати на Aphid і встановити такі властивості:

- Встановіть його властивість "Зображення" на завантажений вами файл із зображенням попелиці.
- Встановіть його властивість Interval рівною 10, щоб, як і сонечко, він рухався кожні 10 мілісекунд.
- Встановіть його швидкість на 2, щоб він рухався не надто швидко, щоб сонечко могло його зловити.

Не турбуйтеся про його властивості x і y (якщо він не знаходиться на вершині сонечка) або властивість Heading, які будуть встановлені в редакторі блоків.

Боротьба з попелиць

Експериментуючи, ми виявили, що для попелиці найкраще змінювати напрямок руху приблизно раз на 50 мілісекунд (5 "тиків" Clock1). Одним з підходів до забезпечення такої поведінки може бути створення другого годинника з інтервалом TimerInterval 50 мілісекунд. Однак ми б хотіли, щоб ви спробували інший метод, щоб ви могли дізнатися про блок випадкового дробу, який повертає випадкове число більше або дорівнює 0 і менше 1 при кожному виклику. Створіть процедуру UpdateAphid, як показано на *рисунку 5-11*, і додайте її виклик у Clock1.Timer.

Як працюють блоки

Щоразу, коли спрацьовує таймер (100 разів на секунду), викликаються як UpdateLadybug (як і раніше), так і UpdateAphid. Перше, що відбувається у UpdateAphid, це генерування випадкового числа від 0 до 1 - наприклад, 0.15. Якщо це число менше 0.20 (що трапляється у 20% випадків), попелиця змінить свій напрямок на випадкову кількість градусів між 0 і 360. Якщо число не менше ніж 0,20 (що буде мати місце в решту 80% часу), попелиця не змінить свого курсу.

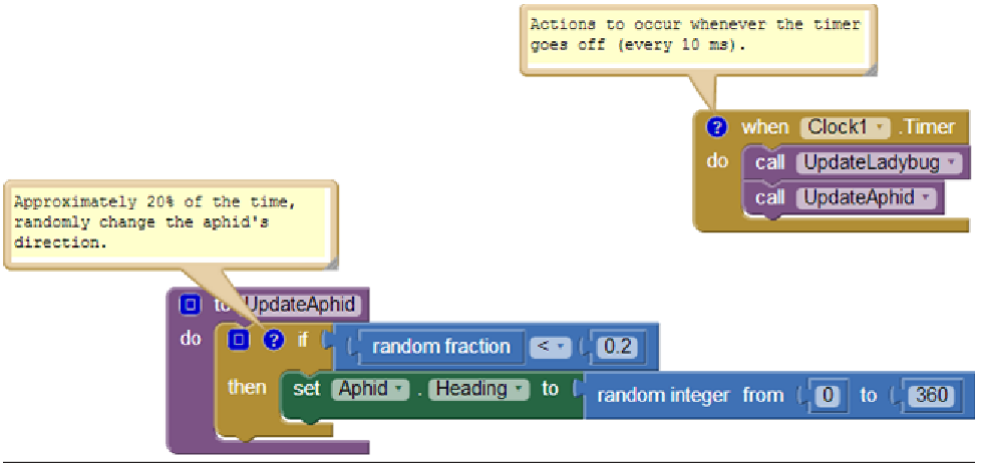


Рисунок 5-12. Додавання процедури UpdateAphid

ЗАПРОГРАМУВАВШИ СОНЕЧКО З'ЇСТИ ПОПЕЛИЦЮ

Наступним кроком є налаштування сонечка так, щоб воно "з'їдало" попелицю, коли вони зіткнуться. На щастя, App Inventor надає блоки для виявлення зіткнень між компонентами `ImageSprite`, що викликає питання: що має статися при зіткненні сонечка і попелиці? Можливо, ви захочете зупинитися і подумати про це, перш ніж читати далі.

Щоб розібратися з тим, що відбувається при зіткненні сонечка та попелиці, створимо процедуру `EatAphid`, яка виконує наступні дії:

- Підвищує рівень енергії на 50, імітуючи поїдання смаколика.
- Спричиняє зникнення попелиці (встановивши її властивість `Visible` у false).
- Змушує попелицю припинити рух (встановивши її властивість `Enabled` у false).
- Змушує попелицю переміститися у випадкове місце на екрані. (Це відбувається за тією ж схемою, що і код для переміщення крота у MoleMash).

Перевірте, щоб ваші блоки відповідали *малюнку 5-12*. Якщо у вас є інші ідеї щодо того, що має відбуватися, наприклад, звукові ефекти, ви можете додати їх також.

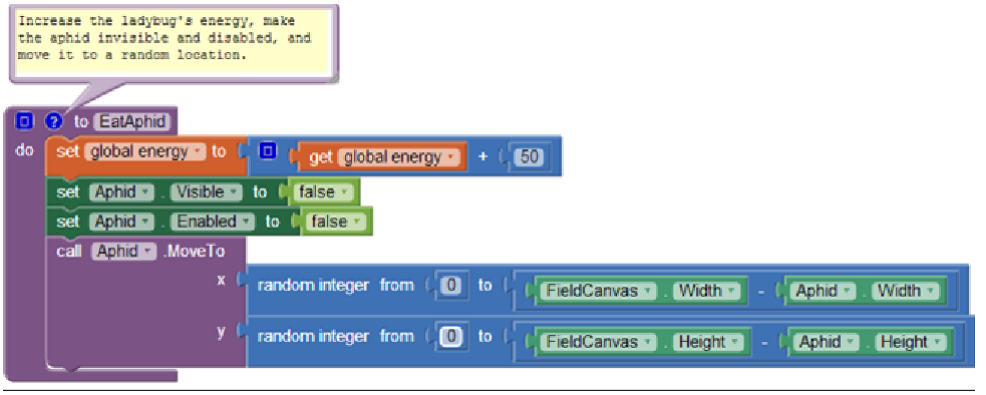


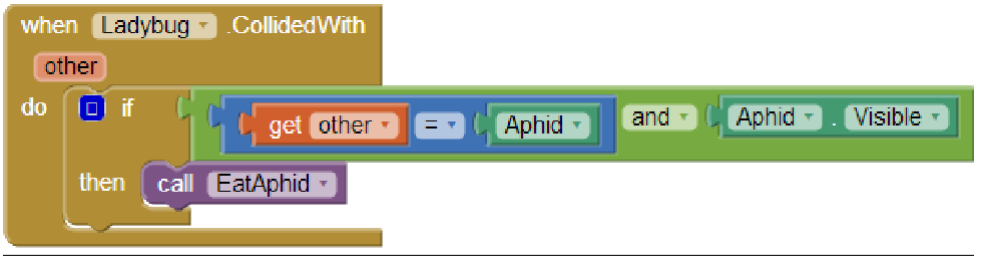
Рисунок 5-13. Додавання процедури EatAphid

Як працюють блоки

Кожного разу, коли викликається `EatAphid`, вона додає 50 до змінної енергії, запобігаючи голодній смерті сонечка. Далі, властивості `Visible` та `Enabled` попелиці встановлюються у false, щоб попелиця зникала і припиняла рух відповідно. Нарешті, для виклику `Aphid.MoveTo` генеруються випадкові координати x та y, щоб коли попелиця з'явиться знову, вона була в новому місці (інакше її з'їли б, як тільки вона з'явилася).

ВИЯВЛЕННЯ ЗІТКНЕННЯ СОНЕЧКА ТА ПОПЕЛИЦІ

На *рисунку 5-13* показано код для виявлення зіткнень між сонечком та попелицею.



Малюнок 5-14. Виявлення зіткнень між сонечком і попелицею та реагування на них

Як працюють блоки

Коли сонечко стикається з іншим `ImageSprite`, викликається `Ladybug.CollidedWith`, з параметром "other", прив'язаним до того, з чим зіткнулося сонечко. Наразі, єдине, з чим він може зіткнутися - це попелиця, але пізніше ми додамо жабу. Ми будемо використовувати *захисне програмування* і явно перевіряти, що зіткнення відбулося з попелицею, перш ніж викликати `EatAphid`. Також ми перевіримо, чи видима попелиця.

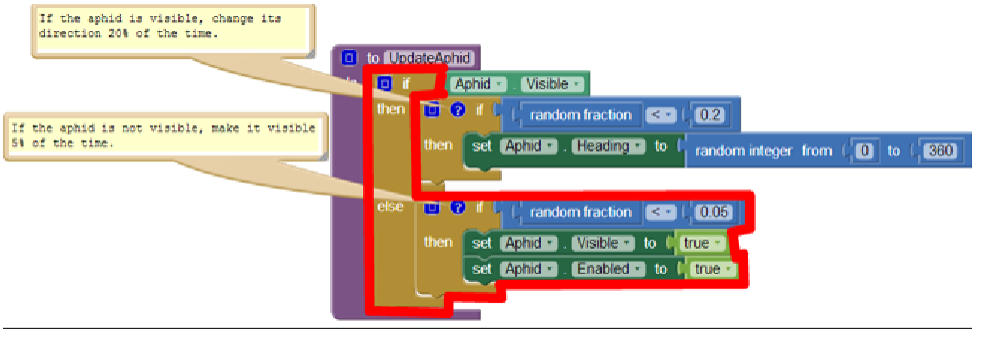
В іншому випадку, після того, як попелиця буде з'їдена, але до того, як вона знову з'явиться, вона може зіткнутися з

сонечко знову. Без перевірки невидима попелиця була б знову з'їдена, що спричинило б черговий стрибок енергії, і користувач не зрозумів би, чому.

Примітка *Захисне програмування - це практика написання коду таким чином, щоб він все одно працював, навіть якщо програму буде змінено. На рисунку 5-13 перевірка "other = Aphid" не є обов'язковою, оскільки єдине, з чим може зіткнутися сонечко, - це попелиця, але наявність цієї перевірки запобіжить збоям у роботі нашої програми, якщо ми додамо ще один* `ImageSprite` *і забудемо змінити* `Ladybug.CollidedWith`. *Програмісти зазвичай витрачають більше часу на виправлення помилок (програмних, а не тих, що поїдають попелиць), ніж на написання нового коду, тому варто витратити трохи часу на написання коду таким чином, щоб запобігти виникненню проблем в першу чергу.*

ПОВЕРНЕННЯ ПОПЕЛИЦІ

Щоб попелиця зрештою з'явилася знову, вам слід змінити `UpdateAphid`, як показано на *Рисунок 5-14*, так, щоб він змінював напрямок руху попелиці, тільки якщо її видно. (Якщо тлю не видно (наприклад, її нещодавно з'їли), існує ймовірність 1 до 20 (5%), що її буде знову увімкнено - іншими словами, вона знову стане придатною для поїдання.



Малюнок 5-15. Модифікація UpdateAphid, щоб повернути до життя невидиму попелицю

Як працюють блоки
`UpdateAphid` стає досить складним, тому давайте уважно розглянемо його поведінку:

- Якщо попелицю видно (а так воно і буде, якщо тільки її не з'їли), `UpdateAphid` поводиться так, як ми писали спочатку. Зокрема, існує 20% ймовірність того, що він змінить свій напрямок.

- Якщо попелицю не видно (тобто її нещодавно з'їли), то виконується частина `else` блоку `if else`. Потім генерується випадкове число. Якщо воно менше 0.05 (а це буде в 5% випадків), попелиця знову стає видимою і активується, що дає їй можливість бути з'їденою знову.

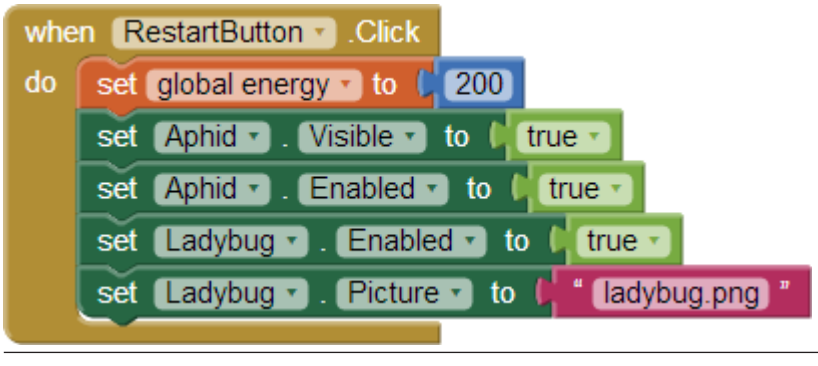
Оскільки `UpdateAphid` викликається `Clock1.Timer`, що відбувається кожні 10 мілісекунд, а ймовірність того, що попелиця знову стане видимою, становить 1 з 20 (5%), їй знадобиться в середньому 200 мілісекунд (1/5 секунди), щоб знову з'явитися.

ДОДАВАННЯ КНОПКИ ПЕРЕЗАПУСКУ

Як ви могли помітити, тестуючи додаток з новою функцією поїдання попелиць, грі дуже потрібна кнопка перезапуску. (Це ще одна причина, чому корисно проектувати і створювати додаток невеликими частинами, а потім тестувати його - ви часто виявляєте речі, які ви пропустили, і легше додати їх під час роботи, ніж повертатися і змінювати їх після завершення програми). У Дизайнері компонентів додайте компонент `Button` під `EnergyCanvas`, перейменуйте його на "RestartButton" і встановіть для його властивості `Text` значення "Restart".

У редакторі блоків створіть код, показаний на *рисунку 5-15*, який виконує наступні дії при натисканні кнопки `RestartButton`:

1. Встановіть рівень енергії на 200.
2. Увімкніть попелицю і зробіть її видимою.
3. Знову увімкніть сонечко і змініть його зображення на живе (якщо тільки ви не хочете зомбі-сонечка!).



Малюнок 5-16. Перезапуск гри при натисканні кнопки RestartButton

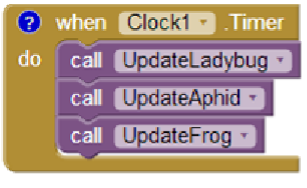
ДОДАВАННЯ ЖАБИ

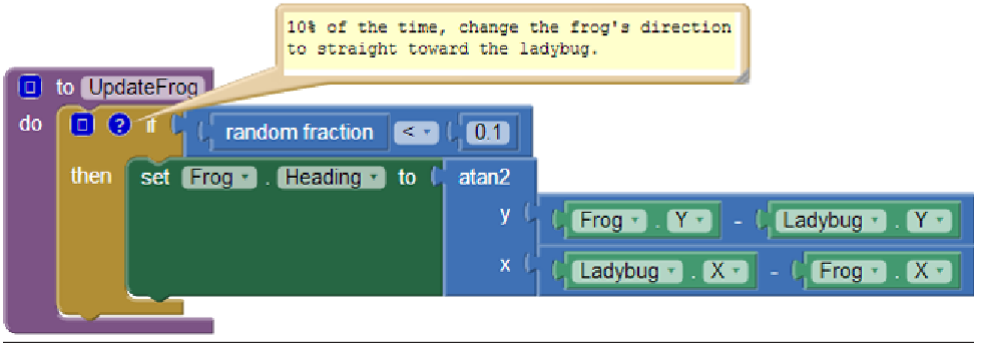
Зараз зберегти сонечко живим не так вже й складно. Нам потрібен хижак. Зокрема, ми додамо жабу, яка рухається прямо до сонечка. Якщо вони зіткнуться, сонечко стане вечерею, і гра закінчиться.

Змусити жабу гнатися за сонечком

Першим кроком до налаштування жаби на переслідування сонечка є повернення до Конструктора Компонентів і додавання третього ImageSprite-жаби до `FieldCanvas`. Встановіть його властивість `Picture` на відповідне зображення, властивість `Interval` на `10`, а властивість `Speed` на `1`, оскільки жаба має рухатися повільніше, ніж інші істоти.

На рисунку 5-16 показано `UpdateFrog`, нову процедуру, яку слід створити і викликати з неї Годинник `1`. Таймер.





Малюнок 5-17. Змушуємо жабу рухатися до сонечка

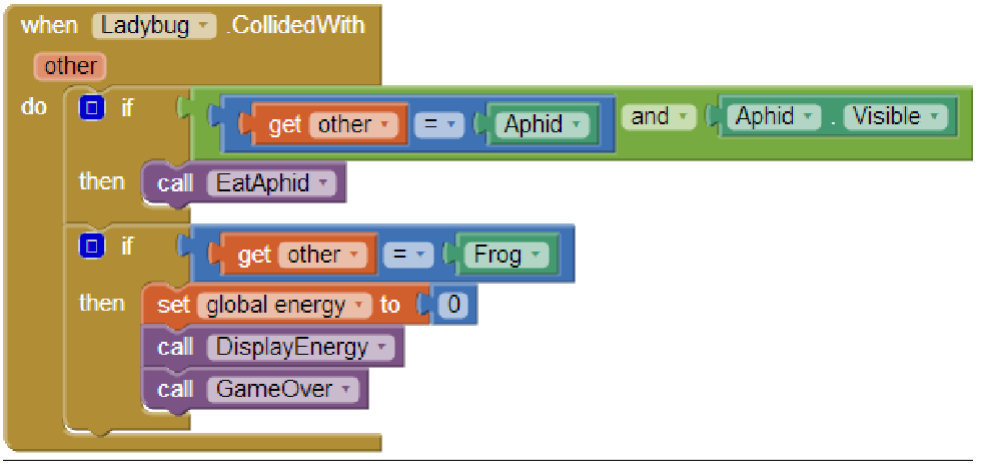
Як працюють блоки

Ви вже повинні бути знайомі з використанням блоку `випадкових дробів для того, щоб` зробити так, щоб подія відбулася з певною ймовірністю. У цьому випадку існує 10% ймовірність того, що жаба змінить напрямок і попрямує прямо до сонечка. Для цього потрібна тригонометрія, але не панікуйте - вам не доведеться розв'язувати її самостійно. App Inventor обробляє безліч математичних функцій за вас, навіть такі, як тригонометрія. У цьому випадку ви хочете використати блок `atan2` (арктангенс), який повертає кут, що відповідає заданому набору значень x і y. (Для тих, хто знайомий з тригонометрією, причина

аргумент y у `atan2` має протилежний знак, ніж ви очікували - протилежний порядок аргументів для віднімання - це означає, що координата y збільшується у напрямку вниз на Android Canvas, що є протилежним до того, що відбувалося б у стандартній системі координат x-y).

Підготуйте жабу з'їсти сонечко

Тепер нам потрібно модифікувати код зіткнення так, щоб при зіткненні сонечка з жабою рівень енергії та індикатор падали до 0 і гра закінчувалася, як показано на *Рисунку 5-17*.



Малюнок 5-18. Змушуємо жабу з'їсти сонечко

Як працюють блоки

На додаток до першого `if`, який перевіряє, чи не зіткнулося сонечко з попелицею, тепер є другий `if`, який перевіряє, чи не зіткнулося сонечко з жабою. Якщо сонечко і жаба зіткнулися, то відбуваються три речі:

1. Змінна енергія падає до 0, тому що сонечко втратило свою життєву силу.
2. `DisplayEnergy` викликається, щоб стерти попередню енергетичну лінію (і намалювати нову, порожню).
3. Процедура `GameOver`, про яку ви писали раніше, покликана зупинити сонечко і змінити його зображення на зображення мертвого сонечка.

ПОВЕРНЕННЯ СОНЕЧКА

`RestartButton.Click` вже містить код для заміни зображення мертвої божої корівки на зображення живої божої корівки. Тепер вам потрібно додати код для переміщення живого сонечка у довільне місце. (Подумайте, що станеться, якщо ви не перемістите

сонечко на початку нової гри. Де вона буде знаходитися по відношенню до жаби?) На *рисунку 5-18* показано блоки для переміщення сонечка при перезапуску гри.

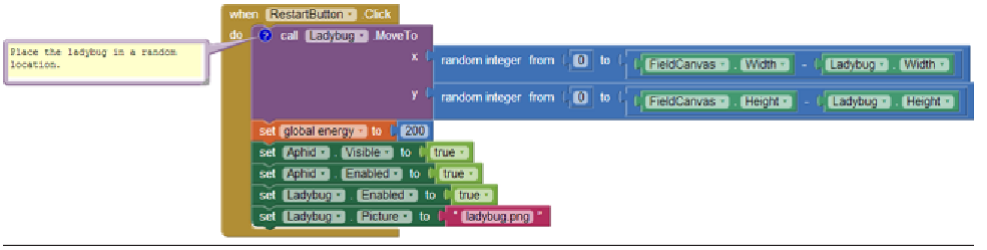


Рисунок 5-19. Фінальна версія RestartButton.Click

Як працюють блоки

Єдина відмінність цієї версії `RestartButton.Click` від попередньої - блок `Ladybug.MoveTo` та його аргументи. Вбудована функція `random integer` викликається двічі: один раз для генерації легальної координати x і один раз для генерації легальної координати y. Хоча ніщо не заважає сонечку опинитися поверх попелиці або жаби, шанси протилежні.

Перезапустіть гру і переконайтеся, що сонечко з'являється у новому випадковому місці.

ДОДАВАННЯ ЗВУКОВИХ ЕФЕКТІВ

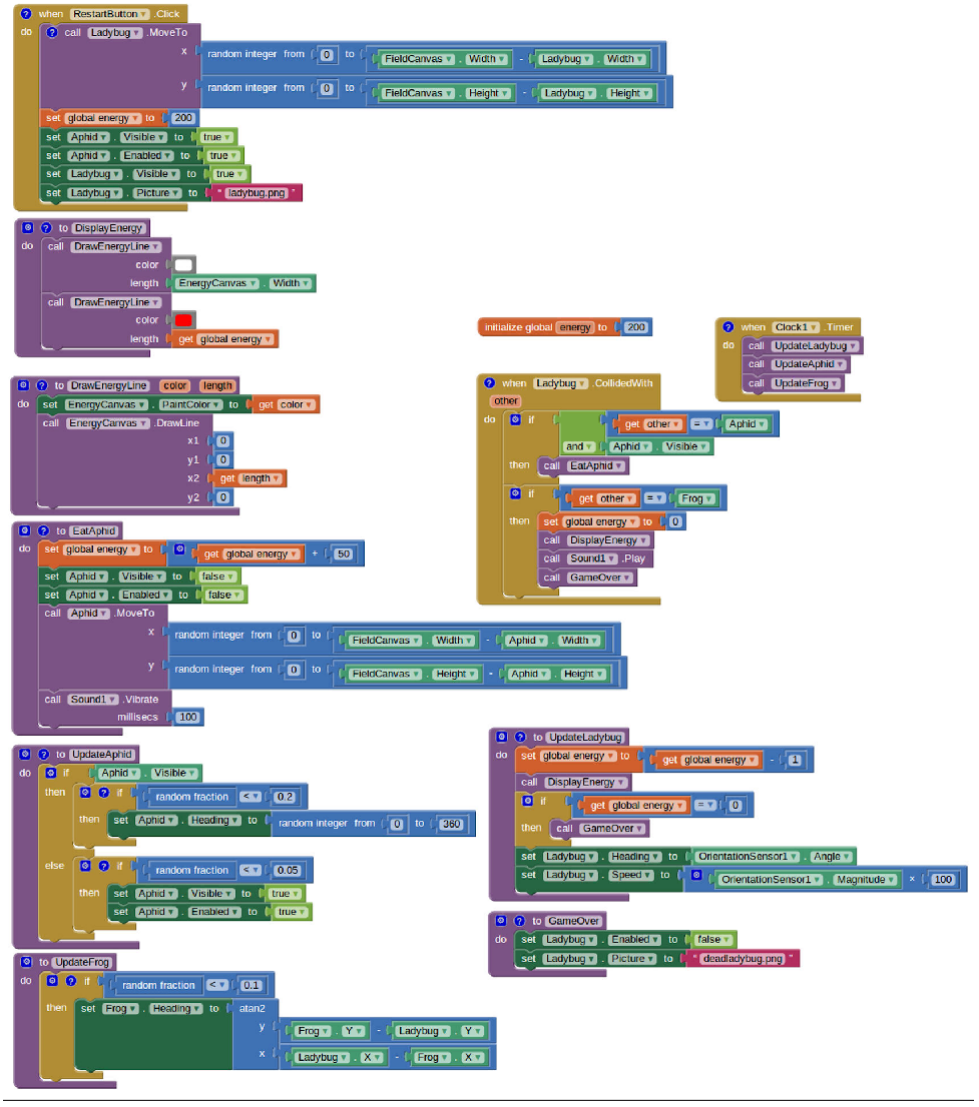
Коли ви тестували гру, ви могли помітити, що немає дуже хорошого зворотного зв'язку, коли щось з'їдається. Щоб додати звукові ефекти і тактильний зворотній зв'язок, зробіть наступне:

1. У Дизайнері компонентів додайте компонент "`Звук`". Встановіть `джерело` звуку на завантажений вами звуковий файл.
2. Перейдіть до редактора блоків, де ви це зробите:

- Зробіть так, щоб пристрій вібрував, коли попелиця з'їдає попелицю, додавши `Sound1.Вібраційний` блок з аргументом 100 (мілісекунд) у `EatAphid`.
- Зробіть так, щоб жаба з'їла сонечко, додавши виклик до `Звук 1. Запустити` у програмі `Ladybug.CollidedWith` безпосередньо перед викликом `GameOver`.

Повний додаток: Погоня за сонечком

На рисунку 5-19 показано остаточну конфігурацію блоків для Ladybug Chase.



Малюнок 5-20. Повна версія програми Ladybug Chase

Варіації

Ось кілька ідей, як покращити або налаштувати цю гру:

- Наразі жаба та попелиця продовжують рухатися після завершення гри. Попередьте це, встановивши їхні властивості `Enabled` у значення false у `GameOver` та true у `RestartButton.Click`.

- Виведіть на екран рахунок, який показує, як довго сонечко залишилося живим. Ви можете зробити це, створивши мітку, яку ви будете інкрементувати у `Clock1.Timer`.
- Зробіть енергетичну смужку більш помітною, збільшивши висоту `EnergyCanvas` до 2 і намалювавши дві лінії, одну над іншою, у `DrawEnergyLine`. (Це ще одна перевага наявності процедури, а не дубльованого коду для стирання і перемальовування енергетичної лінії: вам просто потрібно внести зміни в одному місці, щоб змінити розмір -або колір, або розташування лінії).
- Додайте атмосфери за допомогою фонового зображення і більше звукових ефектів, наприклад, звуків природи або попередження, коли рівень енергії сонечка стає низьким.
- Чи ускладнюється гра з часом, наприклад, за рахунок збільшення швидкості жаби? або зменшуючи його властивість `Interval`.
- Технічно, сонечко повинно зникати, коли його з'їдає жаба. Змініть гру так, щоб сонечко ставало невидимим, якщо його з'їдає жаба, але не зникало, якщо вона помирає з голоду.
- Замініть зображення сонечка, попелиці та жаби на більш відповідні до вашого смаку, наприклад, хоббіта, орка та злого чарівника або повстанського винищувача, енергетичної капсули та імперського винищувача.

Підсумок

Маючи за плечима дві гри (якщо ви закінчили навчальний посібник з MoleMash), ви тепер знаєте, як створювати власні ігри, що є метою багатьох програмістів-початківців або початківців! Зокрема, ви навчилися:

- Ви можете мати декілька компонентів `ImageSprite` (сонечко, попелиця і жаба) і виявляти колізії між ними.
- `OrientationSensor` може визначати нахил пристрою, і ви можете використовувати це значення для керування рухом спрайта (або будь-чого іншого, що ви можете собі уявити).
- Один компонент `Clock` може керувати кількома подіями, які відбуваються з однаковою частотою (зміна напрямків руху сонечка і жаби) або з різною частотою, за допомогою блоку випадкового дробу. Наприклад, якщо ви хочете, щоб подія відбувалася приблизно одну четверту (25 відсотків) часу, помістіть її в тіло блоку `if`, який виконується

лише тоді, коли результат `випадкового дробу` менший за 0.25.

- Ви можете мати кілька компонентів `Canvas` в одному додатку, що ми і зробили, щоб мати ігрове поле і відображати змінну графічно (а не через мітку).

- Ви можете визначати процедури з параметрами (такими як "колір" і "довжина" у `DrawEnergyLine`), які керують поведінкою, що значно розширює можливості процедурної абстракції.