

Project 1.1.7 Sprite Smash

Introduction

|  |  |
| --- | --- |
| In the game Sprite Smash a sprite pops up at random positions on the screen. The player scores points by tapping the sprite before it jumps to a new location.  You can build this game in App Inventor by applying what you have learned about programming. The example app shown here uses a panda sprite but you will use a sprite that you created in the last activity.  As you have learned, mobile apps are event based. Sprite Smash is driven by an internal event—the clock timer firing, and an external event—the player tapping on the screen. To build this app you will practice what you learned about agile development to break the problem into small pieces and build one bit of the game at a time. The best games aren’t built in a day! |  |

Materials

* Computer with browser
* Android device with AI Companion
* Google ID

Procedure

1. Form pairs as directed by your teacher. Practice a professional greeting by making eye contact, shaking hands, and exchanging positive words about the work you will be doing.
2. Log in to MIT App Inventor and select **Start New Project**. Name the project SpriteSmash according to the naming convention that you have been using for other projects in this class.
3. SpriteSmash consists of six components.
   1. The *Canvas* serves as the field where the sprite jumps around.
   2. The *ImageSprite* is a picture of a sprite that is contained inside the canvas component. (Note: Sprites must be on a canvas.)
   3. The *Clock* is used as a timer to move the sprite around the canvas at regular intervals.
   4. The *Label* is displays the player’s score.
   5. The *Button* resets the game.
4. Drag the components into your app as shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Designer Drawer** | **Purpose** | **Initial Properties** |
| Canvas | Drawing and Animation | Provide a touch-sensitive surface where you can draw and animate |  |
| Image Sprite | Drawing and Animation | Appear in random places, detect when player taps it |  |
| Label | User Interface | Display score | Text = “Score: 0” |
| Button | User Interface | Reset game | Text = “Reset” |
| Clock | Sensors | Fire at regular intervals to drive sprite movement; game timer. | Interval = 500  (milliseconds between timer events) |

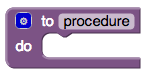
1. Set the initial **properties** of the components.

|  |  |  |
| --- | --- | --- |
| **Component Name** | **Property** | **Value** |
| Canvas1 | width  height | Fill Parent  300 pixels |
| Label1 | Text | Score = 0 |
| Button1 | Text | Reset |
| Clock1 | Interval | 500 milliseconds |
| ImageSprite1 | Image | Set the image source file to the sprite you created in the last activity. Adjust width and height properties to make the image an appropriate size. |
| ImageSprite1 | Height | 50 to 75 pixels (choose appropriate size for your image) |
| ImageSprite1 | Width | 50 to 75 pixels (choose appropriate size for your image) |

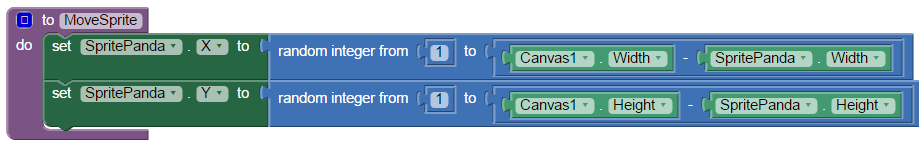
1. Rename each component so that the name describes the component's purpose. You might use the following names.

|  |  |
| --- | --- |
| **Component** | **Name** |
| Button1 | ButtonReset |
| Label1 | LabelScore |
| ImageSprite1 | SpritePanda |

1. Switch to the **Blocks view** by clicking the Blocks button in the upper-right corner to begin programming the behavior of the app.
2. How should the app behave? The most basic functionality is that the sprite should move around the screen.This can be stated as: **When the timer fires, change the sprite’s position on the canvas to a new random location.**
   1. There are several ways to give a sprite a new location on a canvas. One method is to assign a new x, y position on the coordinate plane of the canvas. This repositioning will happen many times during the game so it is a good idea to modularize these steps by putting them into a **procedure**.
   2. A procedure is a block of code that is called by a single name. Defining a procedure allows you to repeat a sequence of steps by simply calling the name of the procedure. Drag out a to procedure do block from the Procedure drawer.



* 1. Give the procedure a meaningful name such as MoveSprite and put the statements for the procedure inside. The code should update the sprite’s x position and y position to a random number between the lowest and highest possible values for the x and y. One possible solution is shown in the following figure:



* 1. Consider what would happen if you did not subtract the sprite’s width from the canvas width, and the sprite’s height from the canvas height.

1. When should the sprite move? Look in the clock component’s drawer for the event handler that triggers regularly based on the clock’s interval property. Drag out the event handler and put the call MoveSprite block from Procedure drawer inside the event handler.
2. Connect your device and test your program.
3. Continue only when you are satisfied that you have completed this task. The sprite should be moving around the screen randomly.

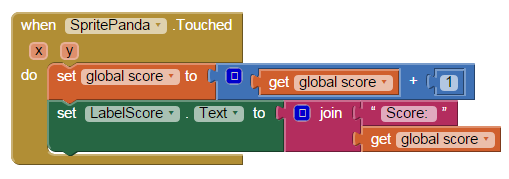
Debugging tips:

* + The clock’s enabled property should be checked and its interval value should be 500 (or less if you want the sprite to move faster). Interval is in milliseconds, 1000 ms = 1 s.
  + Check that the canvas *width* is set to Fill parent and that its height is set to a number of pixels that is less than the screen height (300–400 usually works.)
  + Confirm that the sprite is sized correctly to fit onto the canvas by checking its height and width properties.

1. As you learned in the *CountMeIn* activity, decomposing a problem is an important step in developing a solution. A product backlog and sprint task list for the Sprite Smash game could look like the table below.

|  |  |  |
| --- | --- | --- |
| **Backlog** |  | **Sprint Task List for item #1** |
| * Player should earn a point for tapping the sprite while it is visible on the screen * Player wants to see the score at the bottom of the screen. * Player wants to be able to press the reset button to start the game over. | * When the player taps the sprite, increase the score and display the score on the screen. |

1. When the player taps the sprite, increment the score and display it on the screen.  
   1. Look in the sprite component’s drawer for the event handler that triggers when the sprite is touched. Drag out the event handler.
   2. What steps should happen when this event handler is triggered?
      * Increase the score by 1.
      * Update the LabelScore text property to display the new score.
   3. Where can we keep track of a score? Time to declare a variable!
      * Define a new **global variable** for the score and initialize it to 0.
   4. Put the following blocks inside the when Sprite.Touched event handler:
      * + set score = score + 1
        + set LabelScore.Text = join (“Score:”, score)



1. Stop and test your app. Continue only when you are sure you have accomplished the sprint task. When you tap on the sprite, the score should increase by 1 and the label should display the new score.
2. When the button is pressed, reset the score to 0 and show Score: 0 on the screen.

This functionality is pretty simple for you because you already learned how to program a button when you built the *CountMeIn* app. Get the click event handler for the ButtonReset component and put the appropriate blocks inside of it. What should happen when the Reset button is clicked?

* + - set score = 0
    - set LabelScore.Text = “Score: 0”

1. Test your game. Continue only when you are satisfied that you have completed the third task. Pressing the reset button should cause the score to reset to 0 and the label on the screen should update correctly.
2. Improve your game by adding at least one of these features to your app.
   1. Increase the speed of the sprite as the game goes on.
   2. Count the number of misses. (Hint: Use the when Canvas.Touched event.)
   3. Add another sprite as a diversion. Take points away if the wrong sprite is tapped.
   4. Limit the length of the game. Record and display a high score that persists between games.
3. Save your work in App Inventor. Provide evidence of your work to your teacher, as directed. Some methods for sharing your work with your teacher:
   1. Show your App Inventor screen.
   2. Demonstrate the app on a tablet using AI Companion.
   3. Download the App Inventor program by selecting **Projects** > **Export selected project (.AIA) to my computer**).
   4. Download the Android app by selecting **Build** > **App (save .APK to my computer)**.

Conclusion Questions

1. What is a global variable?
2. What are some of the advantages gained by using a procedure in a program?