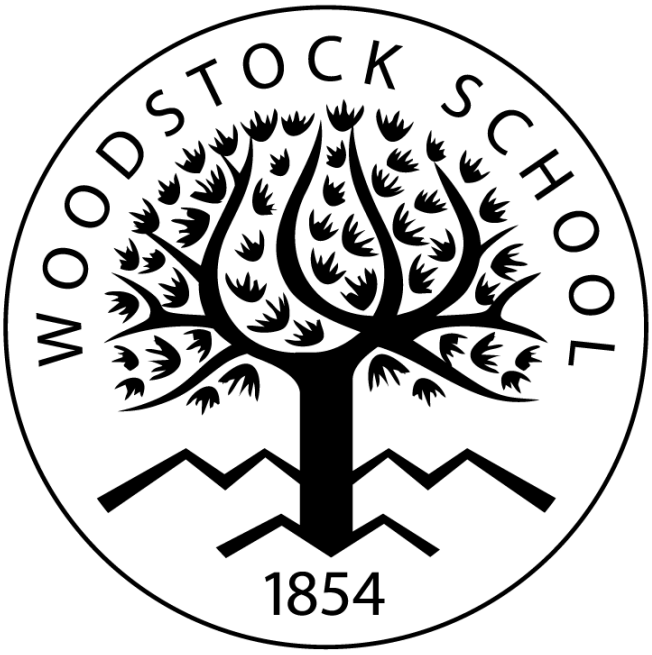
Building a Game

AP Computer Science Principles



Name: _____

Contents

Activity #1

Introduction

In this activity, you will be programming the game logic necessary to click on our Processing Bee and respond accordingly. The area of the screen reprsenting a particular game components clickable area is generally known as that component's "hit box". Although these hit boxes can be very specific, conforming exactly to a particular component's dimensions, the earliest and easiest hit boxes are just that: rectangular boxes representing where you could click on, or "hit", a particular object. You will be programming this simple form of a hit box by identifying the specific location and dimensions of this hit box for a randomly placed bee, identifying whether or not the mouse was clicked within that box, and responding accordingly.

Exercises

1. Implement a program to draw a Processing Bee in a random location on the screen.
   Note: Because you will need to know the location of the bee, you should keep track of the `x` and `y` coordinates for the bee in global variables.
2. Implement the `checkHit()` method which will be triggered from the `mouseClicked()` method. It should return `true` if the location of the mouse click represents a "hit" in accordance to the current bee's calculated hit box and `false` otherwise.
   Note: It may be helpful to temporarily draw the hit box for each bee in order to accurately test this method.
3. Have your program respond to correctly hitting a bee by clearing the old bee and generating a new one. This process should continue as long as the program is running.

Questions

Question #1: Why is it important to separate the program logic of checking for a "hit" from the logic of drawing or even responding to a mouse click?

Question #2: How would you modify your program to account for more than one clickable bee on the screen at a given time?

Question #3: How would you modify your program to allow the bee(s) to move around the screen while remaining clickable?

Activity #2

Introduction

In this activity, you will be introducing game logic allowing for an actual game to be played! In particular: you will introduce a way to keep track of player score, a timer for ending the game, and a timer for replacing the bee with a new one without the user clicking on it.

Exercise

Add each of the following features to your "Whack-a-Bee" game:

1. A tracked score. The user should score points each time he/she successfully clicks on the bee.
2. A countdown timer. The game should enter a "Game Over" state as soon as the countdown timer finishes.
3. A bee timer. The bee should disappear after a set amount of time without being clicked on. This timer should not be displayed to the user.

Questions

Question #4: Another way to implement a "Game Over" state is to have a set number of *lives* for the player, with the player losing a life every time the bee fails to be clicked on before disappearing. Explain how the implementation of this type of feature differs from the one we created in this activity.

Question #5: In what ways could you introduce levels of difficulty into your game? How would you implement them in Processing?

Activity #3

Introduction

In this activity, you will be presented with a range of options to choose from to enhance your game. Each of these options contributes to a more polished gaming experience and a more impressive end result.

Exercise

You must select *thirty* (30) points from the following range of options:

5 Points:

- Introduce Game States for "Start State", "Game State" and "Game Over State",
- Add a second clickable bee to the game,
- Add a graphical representation for the bee timer. This should be something that changes about the bee (colour, alpha, size, etc.) and not a text-based timer,
- Add sound effects to your game using the Processing *Sound* library,
- Increase difficulty of the game over time (either based on the timer or the current score) in some way (ex: making the bees dissappear faster, making the bees appear smaller, etc.),
- Introduce a "number of lives" measure for ending the game. This should reduce the number of remaining lives after each missed bee and end the game when the number of lives reaches zero,
- Add a background image to the game using Processing's image tools.

10 Points:

- Make difficulty level selectable as part of a "Start State" portion of your program. Include at least three difficulty levels,
- Introduce movement for each bee. This should include some random element to it to make the movement of the bee less predictable,
- Add cheat codes to your game to accomplish each of the following:
  - pause the game timer,
  - pause the bee timer,
  - reset all timers and scores,
  - add (5) points to the current score

15 points:

- Use an *array* to store the locations of the previous 5 killed bees and display them in some way on the game screen. These "killed bees" should be distinguished in some way from the currently active bee.
- Keep track of the all-time high score for your program and display it on the screen. Store this high score in a file so that it persists between runnings of the program.

Final Analysis

Question #6: Which part of Activity #1 did you find most challenging? How did you overcome this challenge?

Question #7: Which part of Activity #2 did you find most challenging? How did you overcome this challenge?

Question #8: Why did you choose the options you did for Activity #3?

Question #9: What new programming techniques or methods did you learn as part of this project?