Snake++
Arcade Game

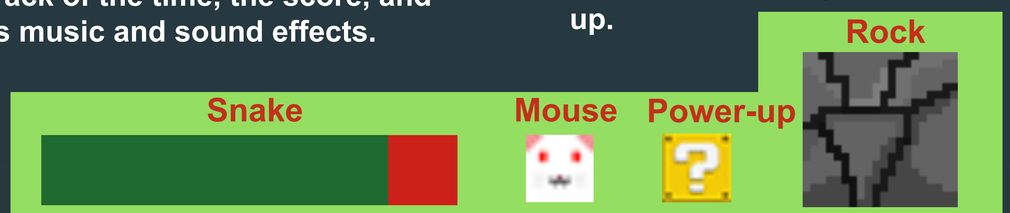
Kevin Li, Anna Staib
Kent Lee, Kiran Tomlinson
Summer Computer Science Institute,
Carleton College, Northfield, MN, USA

Introduction

The goal of our project was to create an arcade version of snake. The classic snake game is a tried and true formula that has been remade over and over in new languages and in new ways. However, we aimed to make an iteration of snake that included power-ups and enemies that were more difficult to eat. Our version of snake is more complex and has more depth, with an added level of difficulty. The programming language we used was python, and we used the Pygame library to code the game mechanics.

Gameplay Features

The snake can be controlled with the arrow keys. Several large and randomly placed rocks serve as obstacles for the snake. If the snake hits a rock, itself, or the edge of the screen, then it's game over. Moving mice spawn on screen, which can be eaten to increase your score and length. For every two mice that are eaten, one power-up spawns on screen, which can be eaten to gain an advantage in the game. The game also keeps track of the time, the score, and includes music and sound effects.



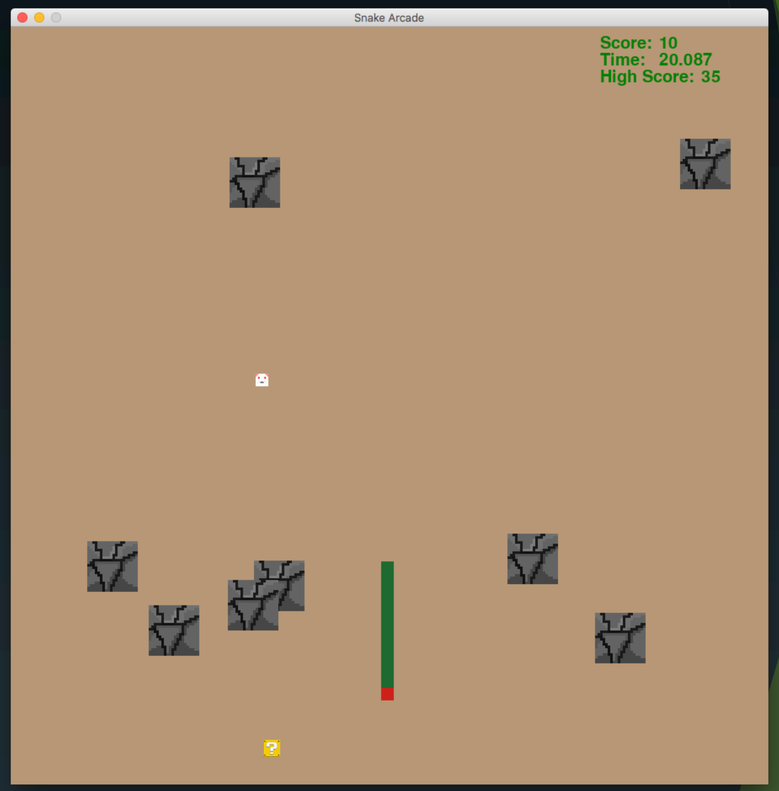
Problems and Solutions

The first problem we had to face was programming the movement of the snake to make the snake body follow the head correctly. To solve this, we coded the snake segments as an array, and had them replace the position of the segment in front of them. The segments all followed the head based on the direction that the head was moving.

Another problem we faced was programming the interactions and collisions between the different objects in the game, such as the snake, segments, mice, rocks, and power-ups. To counteract this, we coded each different object as its own class with a set of methods and variables. We also created sprite groups for them to detect collisions efficiently.

A third problem we had was stopping the mice and power-ups from spawning on top of rocks, as well as stopping rocks from spawning on the snake's initial location. We solved this by detecting collisions before the rest of the game loop, and re-spawning them if they collided. We also kept track of power-ups to make sure there was only one on screen at a time, and counted the number of mice eaten to spawn the next power-up.

Results



Power-ups:

- Rock-eater
- Slowdown
- Speed-boost
- Ghost
- Shed Skin

What We Learned

This project helped us gain a deeper understanding of object oriented programming and using classes. The importance of well written classes was made clear to us, because of how much more efficient the code could be written. Additionally, new features were much easier to implement.

References

https://www.pygame.org/docs/
https://docs.python.org/3/
https://stackoverflow.com/
http://soundbible.com/
https://images.google.com/
https://github.com/

Kent Lee
Kiran Tomlinson