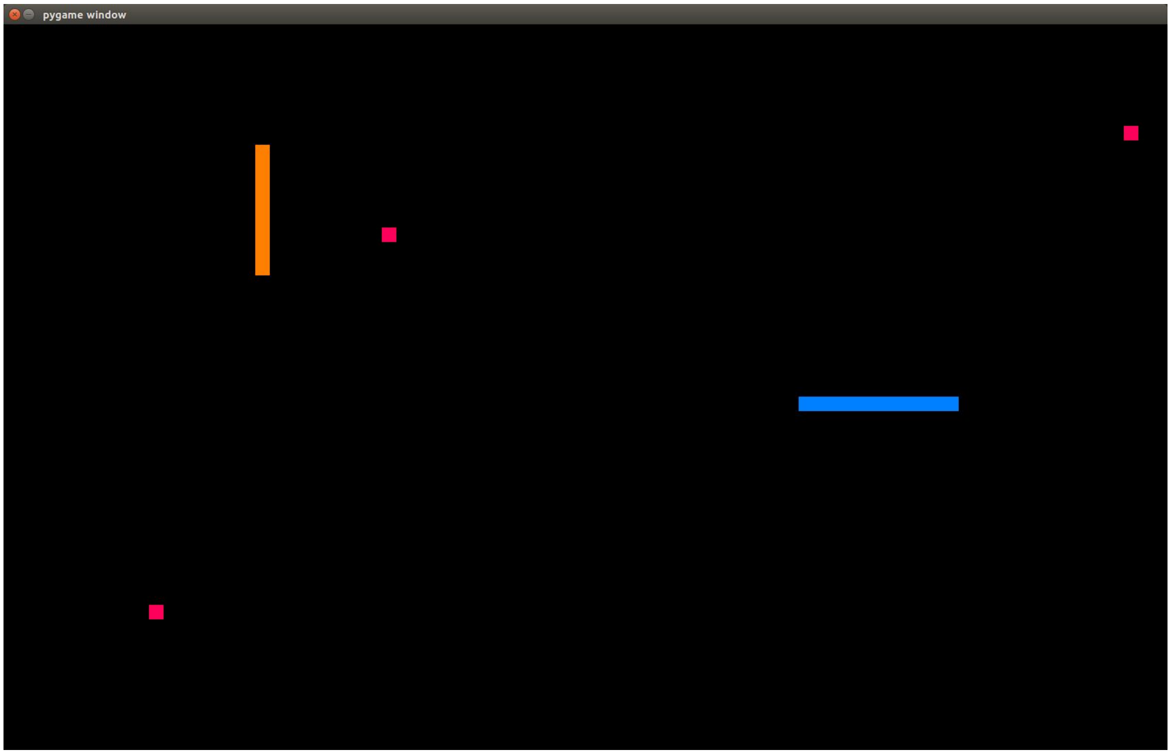
Snakes on Bikes

Eric Elder Jacobsen and Luis Zuñiga

Project Overview

Our project is a combination of 'Tron' and 'Snake' games. Like in 'Snake,' the avatar you control increases in length when you eat an 'Apple,' and like 'Tron,' you lose if the avatar crashes into your opponent, yourself or the wall. We used PyGame to incorporate user input, some basic animations, and collision detection into our game.

Results

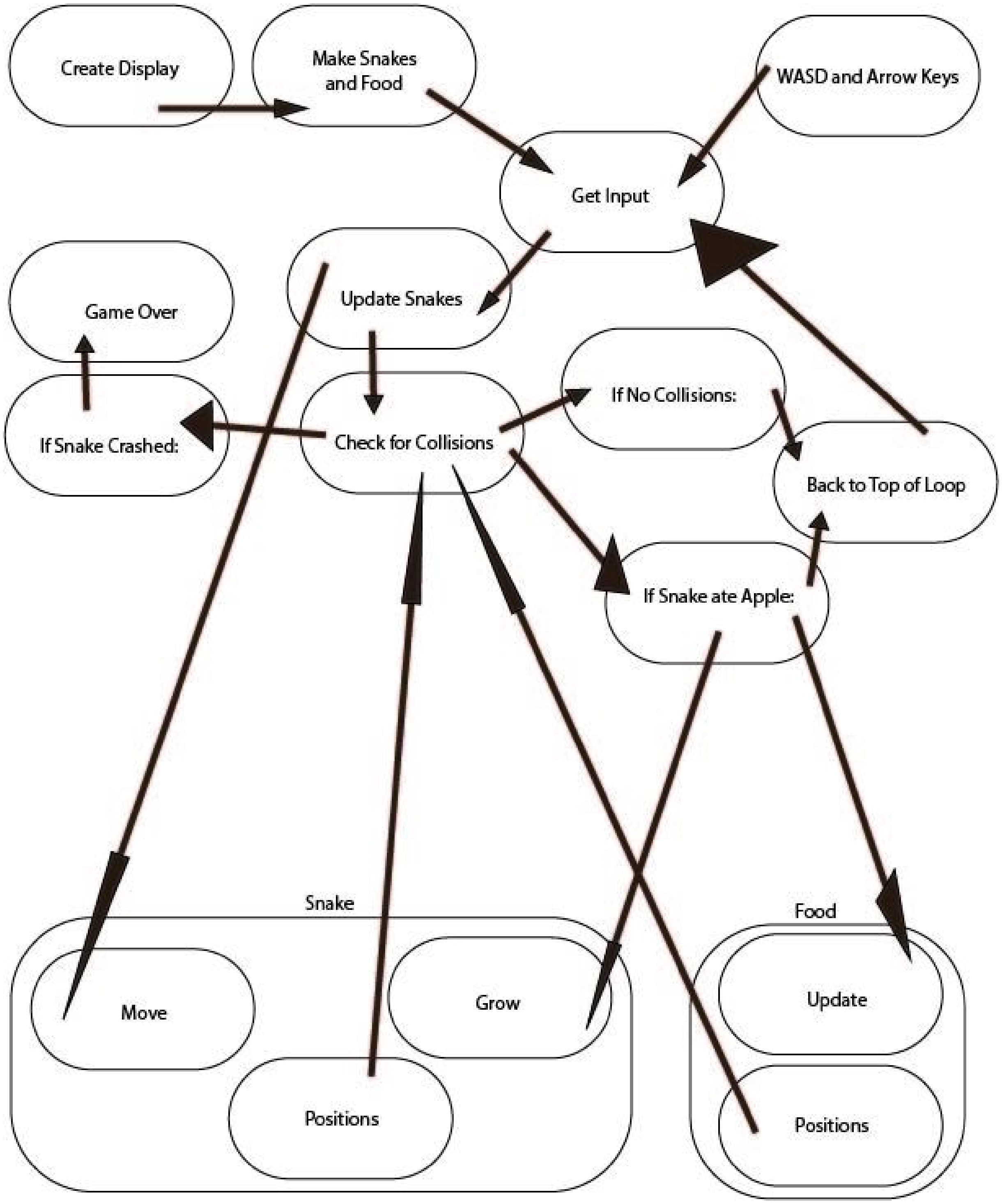


We succeeded in creating a 2-player game that met our minimum requirements. The avatars move, the game can be played between two players, and there is collision detection for obstacles and pick-ups.

The avatars move because of a loop that keeps getting called. The four cardinal directions are given as strings, and the direction is changed depending on which key is pressed. The loop keeps the head of the snake moving in what direction it is facing at the moment, and the tail follows it.

Collision detection was an important part of the game. This allows the game to end when a player collides into something they weren’t supposed to, and also detects when the snakes eat the food. Our collision works between the head of the player and the body of the other player, or the head of the other player and the wall. A game over screen appears when a collision occurs, along with the score (the length of each snake) and a screen that asks if you’d like to play again.

Implementation



Our game consisted of three classes. One was the game itself, which contains instances of the other two classes: snake and food. The food class is the simplest: it is defined only by a location, and has a method that moves it to a random new location on the screen. The snake is defined by a list of positions for all the blocks along its length, the direction that its head is moving, and by a length. It has a move method for each direction, a grow method that adds a block to its length, and an update method that moves the frontmost block in the current direction, and moves each block in the tail to follow the head.

The game class defines the window and then puts two snakes and three foods (called apples) into the blank field. Most of the game occurs in a huge loop. Within the loop, input is collected from the arrow keys for player 1 and the WASD keys for player 2, then both snakes directions and positions are updated accordingly. Using collision detection based on the locations and size of all the blocks in the snake, the game then checks if any blocks have collided. In a snake-apple collision, the snake gets one block longer and the apple is moved to a new location. In a snake-wall or snake-snake collision, it figures out who crashed into who. The person who crashed loses, and the game is over. When the game ends, it exits the loop and displays text on the screen that indicates who won and how long the snakes were when the game ended. It projects this over the images of the snakes and apples, but stops updating the snakes, so they are frozen in place, allowing the players to see the crash. With this text on the screen, the players can press ‘p’ to play a new game, or ‘x’ to quit the program.

Reflection

In my opinion, this project went extremely well. We liked the project and the game topic, but also recognized what we could do within the time frame of the project. In order to gain a better understanding of PyGame and its uses, we approached this project from the ground up. We approached the game by starting with a window, then a window with a block, and then a moving block, then a real snake. The MVP was a product that could be presented as a game itself (Snake Simulator 2018) but also served as a framework for the project we had in mind.

Originally, we had planned to utilize pair programming in order to create the game. However, we found that there was not often a time where were could meet together to work on the code. We started the project separately, but were able to bring together what we worked on to create a first product. The rest of the project mostly followed this model: we would each choose one part of the code to write, and then write it separately before reconnoitering to mesh our codes together. While this worked well for us, I would like to do more true pair-programming next time, just to get an exposure to different strategies and techniques.