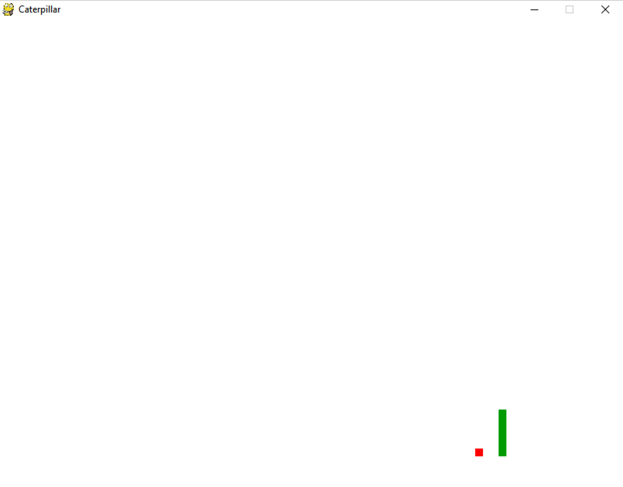
Stephanie  
Professor Harrington  
CS 4233  
25 April 2016

Capstone Recap: Game Programming with Python (Using Pygame)

For the first week of the project, I spent some time refining my ideas and deciding exactly what I wanted to do. I knew I wanted to make a game of some sort, and my research led me to the conclusion that if I wanted to make a 2D game, Python would be better, while for 3D, Java would be better. I had always been interested in Python, so I decided to move forward with game programming in Python with Pygame. I began watching short lectures and reading articles on setting up Python, basic Python programming practices, and getting started with Pygame to familiarize myself with the concepts. One of the lectures I found helped me to create my initial practice game, “Caterpillar.”



“Caterpillar” has boundaries, a dynamic player character, collectibles, and a game over screen that allows you to continue or quit the game.

**Boundaries**

The boundaries in “Caterpillar” exist so that if the player character moves off the screen, the game is over. A game over screen appears prompting the player to either play again or quit. Additionally, if the player character is longer than its original size of one block, it cannot double back on itself without prompting the game over screen. Boundaries were achieved by calculating the size of the window display and setting those as variables that the player character cannot cross. This is represented by the following code block in the main game loop…

http://puu.sh/os5fJ/800e3c8abc.png

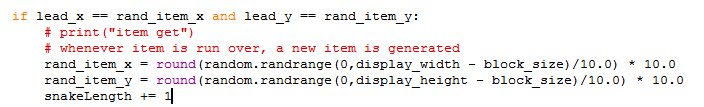
…where lead\_x and lead\_y are the player character’s x and y values.

**Dynamic Player Character**

“Caterpillar’s” player character is dynamic in that its representation changes as the game goes on. As the player character—represented as a simple green line, ranging from a single block to very long, that is supposed to resemble a caterpillar—collects items, it grows in length. This mirrors the classic premise for Snake, where eventually the snake (or caterpillar) will become so long that playing in the game window is impossible without running into itself.

**Collectibles**

Items are generated in a random place on the “Caterpillar” game screen and can be collected by the player character. Items are represented by single red blocks on the game screen. As the character collects items, it grows in length. Each time an item is run over (or “eaten”), a new item is randomly placed on the screen and the player character grows. The following code block achieves this:



**Game Over**

When the player character touches a boundary, including the edges of the game window or its own body, a game over screen is displayed consisting of text and a simple menu. If the user presses the spacebar, they will play the game again. If the user presses the Q key, the game will exit.



After finishing up my practice game, I moved on to starting on my real game, called “Platforming.” I wanted to use some sort of version control software, so I made a GitHub account and practiced a bit using Git to track my changes. To begin with my platforming game, I pulled up some resources on making simple platforming games with Pygame. My biggest challenges for this game involved implementing gravity and a scrolling camera.



“Platforming” is a functional platforming game, though time constraints meant that I could not polish it as much as I would have liked. Features include: image and sound assets, gravity, collide-able objects (platforms), and a scrolling camera. “Platforming” also uses polymorphism to make further development easier. All levels can be subclasses of one superclass Level(), and enemies can be subclasses of one superclass Character().

**Image and Sound Assets**

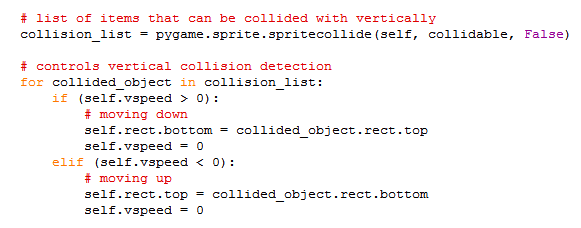
“Platforming” uses images and sounds that are separate from what comes with the Pygame library. The player sprite was created by me and the background image was downloaded from a website that offers royalty-free images specifically for game programming uses (<http://opengameart.org/>). The background music and “jump” sound were downloaded from a royalty-free sounds website (<https://www.freesound.org/>). The jump sound plays whenever the player character jumps, while the background music plays when the player character is drawn to the screen at the start of each level.

**Gravity**

Gravity in “Platforming” works by controlling the vertical and horizontal speed of the character. If the player jumps, they are moved to different coordinates on the game window at a certain speed. Their horizontal and vertical speed reaches zero when they collide with a collide-able object above, below, or on the side of them.

**Collision**

Any object can be programmed with collision detection in “Platforming,” but for now it is only used on platforms. When the player touches one of these platforms, their vertical or horizontal speed reaches zero and they cannot move through the object. This is achieved in several places, but is best exemplified in the following code block:



This block controls only vertical collision. The Pygame module has a built-in functionality to handle sprite collision, called pygame.sprite.spritecollide(), which simplifies implementing collision detection in your game.

**Scrolling Camera**

Perhaps one of the most difficult aspects of “Platforming” was its scrolling camera. The camera follows the player character as they move throughout the level. This works by using a “viewbox” that moves as the character moves and scrolls through the actual world itself. Because of the difficulties of a moving world, I ran into trouble adding boundaries and therefore a “game over” functionality.



As the end of the semester drew close, I ran out of time and was unable to add three features: boundaries, enemies, and game over. The boundaries would have worked similar to “Caterpillar,” but the scrolling camera added some complexity to this that I could not work through in time. The enemies would have been subclassed from the Character() superclass and would have collision detection so that whenever the player touched them, they would die. The game over function would have been a simple menu, also similar to that in “Caterpillar,” that would have displayed on the game window when the player reached a boundary or touched an enemy. As a pet project, it would not be too difficult to add the features mentioned previously as well as others, such as hit points, collectible items, and a high score.

Overall, I learned Python, the fundamentals of game programming, and ended up making a functional game, even if it is not as complete as I would have liked. I also learned how easily plans for programming projects can go awry, and how much work can actually be done by a single person in a set amount of time.