Calvin Makelky

A01872013

Project Report

Frogger in Pygame

Using Python and Pygame I created a simplified version of the classic game Frogger. All of the code was put together by only myself. I relied heavily on the Pygame documentation online to find solutions to the various problems I encountered in making the game. Implementation wise I sat down and drew up my plan in order to simplify big chunks of code into smaller and more attainable tasks. In the end, I really learned how to properly use sprites and how to make a game that constantly changes and can suddenly go back to the starting point.

I chose to create a simplified version of Frogger in Pygame. What inspired me to choose this was that Frogger is a childhood favorite game of mine, and I love playing games. Since we learned collision detection in past assignments, and the fact Frogger is all about avoiding collisions, it seemed like a natural build off from what we have already learned. Frogger is a game where the objective is to get the frog to the top of the screen without being hit by any obstacles. The frog has a limited number of lives, and after each successful round the game resets and the obstacle difficulty increases. The obstacles only move horizontally, while the frog can move up, down, left, or right. The goal is to survive as many rounds as possible. This application is intended for people of all ages that enjoy a simple game.

Pygame is in Python that is primarily used to create games. This application is written to be portable across different operating systems. I did not use in any 3rd party libraries, API’s, or data sources for this application. I relied heavily on the Pygame online documentation and the past homework assignment in Pygame that I completed.

There were many problems that I had to find solutions for. How to create separate sprite classes with each sprite having a certain image displayed over the rectangle. I could not figure out how to get the images on my sprites in the past Pygame assignment. How to make CPU sprites move horizontally while the user sprite is controlled by the arrow keys How to implement two different backgrounds, one on the bottom half and one on the top. How to implement sprite collisions with the images on the rectangles. How to keep the user sprite from going off the screen. Implementing a reset game code that resets the game when the frog runs out of lives. How to implement the top wall detection so the frog can move on to the next level. Implement a counter for what round the frog is on and how many lives the frog has left. How to automatically increase the speed of CPU sprites as each level increases.

To load images onto sprites I first would need to assign the image jpg to an object. Then I would need to use self.image assigned to pygame.image.load(“object”) to load the image onto the sprite under the \_\_init\_\_(self) function for each sprite class. I got this from the Pygame doc on images and on the Sprite cheatsheet by RPI.edu. To move the CPU sprites horizontally I have self.x that controls horizontal movement equal to self.dx so that it moves however fast self.dx is assigned. To detect collisions, one would use the sprite1.rect.colliderect(sprite2) to see if the user sprite (sprite 1) collides with the rectangle of sprite 2 based off the Pygame doc on rectangle collision detection. Repeat that for all CPU sprites. In front of all these is an if statement with variable named deaths decremented each collision, and making the screen red when there is a collision. To keep the user sprite from going off screen I have an if statement under each if pressed(key) that is based on if frog sprite x or y is greater or less than the number of pixels on the edge of the screen. This is pressed key for movement will only move when the frog sprite meets this if condition. In order to detect when the frog reaches the top, an if statement in the game loop is needed. It is set so if the y component of the frog sprite ever equals zero it resets all the sprites and increments a variable called round. To increase the speed for the CPU sprites every level, I set the self.dx equal to round. To reset the game when the frog runs out of lives, I first have a variable named lives before the main game loop that is set to three. In the game loop I have an if statement where if the lives variable equals zero it calls a new instance of all the sprites and sets their starting positions and resets lives to three. It also sets the round variable back to one so the speed starts is at entry level speed. I was able to get half of the background one color and the other half another color. However, I was not able to successfully make the water half designed properly where the user sprite would only be safe it jumped onto the sprite logs. So because I could do only the truck on the road half properly, I just kept the background black.

Implementation wise I broke down the larger tasks into smaller ones and set weekly goals. My first goal was to load images onto the user and CPU sprites. Also that first week I made the CPU sprites move horizontally and the user sprite by arrow keys in all four directions. The next week I implemented an invisible wall boundary for all sprites. Also that week I implemented sprite collisions with the user sprite, and the code to reset the game if the user sprite collides with a CPU sprite. The third week I implemented the wall detection for the user sprite so that all sprites would reset when the sprite reached the opposite wall. Then that week I implemented the font with the round and lives tracker. The last week I implemented the increment for the speed of CPU movement by round. I was not able to implement a second half of the game background that was water where the user sprite had to hop onto log sprites else it dies. Coding wise I wrote the globals and intro code first, then the class code in the first week. Then I wrote and updated the movement code in the game loop. Then I added the collision code in the game loop. Then I added the text code at the end of the game loop. Finally, I added the code for a new round and if the frog dies in the game loop, with the increment speed modification in the truck class.

To sum it up, I successfully made a competent and simplified version of the classic game Frogger. The game was very fun to play and make, and I was right to think that the project was attainable based off past homework in class. I was able to properly detail solutions and implement them successfully on nearly everything I said I would reach for in my original proposal. The Python class lectures with aliens, the past Pygame homework, and the Pygame documentation were all critical to making this project work.

References

<http://www.cogsci.rpi.edu/~destem/gamedev/pygame.pdf>

<http://www.pygame.org/docs/ref/rect.html>

<http://www.pygame.org/docs/ref/sprite.html>

<http://www.pygame.org/docs/ref/image.html>

<http://www.pygame.org/docs/ref/time.html>

<http://www.pygame.org/docs/ref/pygame.html>