Aliisa Roe

CSCI 1310- Boese

Project1

Pangolin Hustle

My program is a game, using pygame, where the player as a pangolin image tries to reach the top of the screen without colliding with any of the moving snake sprites using keyboard arrow keys. I will walk through the code from top to bottom for the most clarity.

First, I attribute all of my graphics, audio, and influential code in comments at the beginning. All sources were intended to be shared online. Next, I had to import libraries and initialize pygame. Then I go on to define my sprite classes in order to animate moving snakes.

In order to create objects that held variables specific to each sprite, I used code to help teach myself how to crop from a single spritesheet I found online and follow it with my own sprite class to hold variables, such as each individual sprite/row of sprites coordinates.

Skipping to the main function, I will explain how these classes are used. First I call a function of print statements that explain the game in the terminal for the game will begin immediately once the print statements are printed and the screen is loaded. In order to let the player read the statements, time.sleep is called from the imported time library.

Next I begin to create my graphics. First, I use the spritesheet class to load the spritesheet. I use a function of the spritesheet class to select individual sections of the spritesheet and list them into an array.

To set up the board, I make another list that stores the previous list drawn at different y coordinates, to create rows of the same image. To shrink my code I put this in a loop, rather than typing out 13 lines of code, and while the y cords were regularly incremented for rows, the x coordinate was selected randomly from the imported random library, so each row would have its graphic beginning in a different place, to make the game more challenging. The array of rows or sprites to make up the game is set up for enemy graphics. Finally I load the pangolin image for the player to move.

Finally, moving to the main game loop, some basic coordinates have to be initialized for use, so they will be ready for regular updating within the loop. First I draw the player’s pangolin image to the x and y coordinates that will be incremented by the arrow keys. The pangolin position is also regularly updated for later checking for collision with snake sprites.

The following for loop looks at the user input, as pygame.events. The game checks for 3 ways to quit—the upper-right x, or the ‘escape’ and ‘q’ keyboard keys. If these are pressed the game loop exits and the program is closed. The only other way to exit the loop is to win or lose.

Still looking at pygame.events, the player uses arrow keyboard presses to move up the screen and avoid snakes. Keyboard presses update the pangolin position.

The next if statement increments an integer to update each row of snake sprites which individual snake sprite image to draw to the screen. Every row will have the same motion in different, randomly generated, x coordinates.

The next for loop will update each row of snake sprites, so the entire screen is moving in animation. This loop uses row class functions check each row’s position independently, having it change direction if it hits the edges of the screen. Though its direction changes independently, the increment of motion is consistent on all rows, whether moving forward or backward by 1 and it’s changed position is saved as a row function, so each row’s coordinates are accessible through the row class.

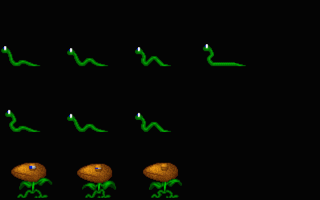
Next, each updated row position is drawn to the screen and compared to the pangolin player’s position. If they collide, using pygame’s colliderect() function, I signal that the pangolin has been hit and exit the loop going through snake rows.

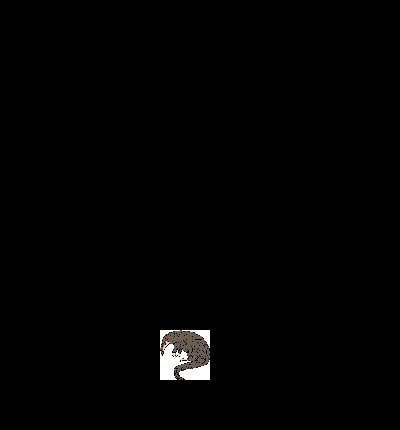
Outside this loop I update game scores, according to whether or not there has been a collision since each row has updated and the pangolin player has moved. There are 10 lives, (multiple are often taken by not being able to retreat quickly) and the remaining lives are printed in the terminal. There are sounds effects for a hit, as well as a stressful red screen that makes the user press arrow keys rapidly for escape as they see.

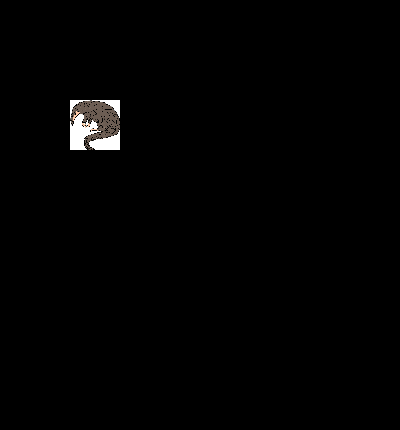
If the player makes it to the top, the screen turns green, prints in the terminal and exits the game loop to close. If the player loses all 10 lives, the screen remains red, prints in the terminal and exits the game loop to close.

The loop continually flips the display to update, during the animation and clock.tick() keeps frames per second consistent on different hardware devices to keep the game at a reasonable speed.

The main function is called to play.







The screenshots are not showing my blitted snake sprites—I promise they are there! I attached these anyway, and the spritesheet up above so that you may infer some of the graphics.