Writeup

Section 1: Your First SFML Project (Window Setup)

In Section 1, I began by initializing the game window using the sf::RenderWindow method. The window size was set at 1800x1600, chosen arbitrarily as a default size for the game. I decided to handle window events in SFML, such as closing the window, by utilizing the default Closed event from SFML events triggered when the user clicks the quit option on the title bar. For resizing events, I made use of the default Resized event to maintain proportional scaling.

Section 2: Drawing Objects

In Section 2, I focused on rendering objects on the screen using the RectangleShape class. This choice allowed me to concentrate on vital game mechanics like gravity, collision, and movement. To move the platform, I decided to match its speed with that of the player. I experimented with different speeds for the platform, but aligning it with the player's speed yielded the best results.

To implement the motion of the moving platform, I utilized two conditional statements and defined boundaries to keep it moving predictably between these limits. I also reversed the platform's velocity each time it reached one of these boundaries, creating a seamless back-and-forth motion. Both platforms received textures, sourced from a free asset, and a green outline to give them a grassy appearance. I intentionally placed the moving platform on a different level than the stationary one to introduce gameplay complexity and to explore gravity and jump mechanics. The player character was given a red color for visibility and thematic coherence. Rendering of objects was accomplished using the .draw method from SFML's window graphics class.

Section 3: Handling Inputs

In Section 3, I addressed input handling, using the built-in SFML isKeyPressed method from sf::Keyboard. The player character's movement was limited to left and right directions due to the presence of gravity and the ability to jump. I adopted standard key bindings: 'A' for left, 'D' for right, and the spacebar for jumping. Additionally, I assigned the 'E' key for constant scaling, allowing the window to be scaled proportionally when the 'E' key is pressed in conjunction with the resize button.

Section 4: Collision Detection

Section 4 outlines the challenges I encountered during the implementation of collision detection. Initially, I devised a coordinate-based collision detection system with a specific ground level.

However, when I introduced a moving platform at a different level, this approach became problematic. I then discovered the utility of .getGlobalBounds() and .intersects() methods, which simplified collision detection significantly.

Whenever the player character collided with a platform, I made its vertical velocity zero, effectively stopping gravity's influence, and repositioned the character on top of the platform. I calculated the character's new y position using the formula platformBorder.top - characterBorder.height. To allow for continued jumping, I set the isJumping flag to false since I had made the character's vertical velocity 0.

Additionally, I encountered issues with excessively high gravity values, causing the character to disappear from the screen before collision detection occurred. Therefore, I had to adjust the value of gravity accordingly.

Another challenge I faced was when the character was on a moving platform. Initially, the character failed to move with the platform. Attempts to solve this issue, such as applying the platform's velocity to the character, resulted in the character abruptly stopping when the platform changed direction. Troubleshooting this problem led to various experiments, including different velocities for the character and platform, but none provided a satisfactory solution and created more problems.

Section 5: Scaling

In Section 5, I discussed the implementation of scaling in the game. SFML's events and the resized method were used to trigger resizing. SFML inherently maintains proportional scaling, so I manually defined a view when resizing to achieve constant scaling. This view was set to match the new width and height of the resized window, effectively displaying only the area that fits within the new screen dimensions. To trigger constant scaling, I utilized the 'E' key in conjunction with the resize button.