Harsh Patel
CSC 363

Final Project Documentation:

About:
The project is a Jetpack Joyride clone created in Rust using the Macroquad crate. The game begins when the user selects the start button. The player can be moved up and down by pressing and releasing the space button. The objective of the game is to go as far as you can without running into the red and orange obstacles. The obstacles are generated at random in both a horizontal and vertical orientation as the player progresses through the game. There is a score tracker in the top left, tracking how far the player has traveled for the given instance of the game. There is no definite end, as it is an infinite game much like the real Jetpack Joyride. If the user collides with an obstacle, a fail message will display, along with the distance they traveled. The user must press “S” to restart the game and then select the start button to play again.

Player Struct:
The player struct contains a position and velocity vector of type Vec2, and a boolean to track whether the player is on the ground or not. The type Vec2 was used to store both position and velocity as it has a field for both the x and y. When a player is on the ground their initial velocity is kept at a higher amount, before being lowered as soon as the player hits the air to emulate a minor jump before settling into a uniform speed. The player struct is also responsible for ensure the player does not fall through the ground or go above the height of the ceiling. This is handled by using the screen's height and creating a slight margin so that the player is always within the bounds of the screen. This struct also provides the logic for drawing the player model, which is simply a circle with a rectangle for a body.

Obstacles:
To create obstacles, the obstacle struct and the obstacle enumerated type is needed. The enumerated type holds a Vertical and Horizontal type outlining the two types of obstacles the game is able to create. The obstacle struct contains a position and size vector using Vec2, along with a type using the earlier defined ObstacleType enumeration. The obstacle struct randomly generates the sizes of the obstacles starting from a minimum to a number relative to the screen size. It uses the enumerated type to decide which side should be the longer side. The position of the obstacle is also randomly assigned relative to the screen width and height. The obstacle is constantly redrawn as the player approaches it. The vertical obstacles are drawn in the color red, and the horizontal obstacles are drawn in the color orange.

Game State Struct:
This struct is responsible for creating the environment of the game, including using the above structs. A GameState struct uses a player from the above player struct, a start button from another user-defined struct, a vector list of obstacles, a float to track distance traveled, the fail message string, and two booleans tracking when the game has been started and when the player has failed. The key part of the GameState struct is handled within the update function and draw function. The draw function generates the background and ground, the player model, the score, and uses the tracked values to draw the obstacles, the start button, and fail message. In the update function, the game starting, failing and restarting logic is included within the if-else

branch encompassing the function. This is responsible for calling the player’s update function, and each obstacle's update function. The distance is updated relative to the frame time, to provide a uniform and meaningful score. This function also checks and sees if any obstacle has come in contact with the player at any point by checking the position and size vectors within the respective structs. The vector list of obstacles removes any obstacles that are behind the player to keep memory efficiency. Random number generation is used to decide between the creation of a vertical and horizontal obstacle.

Main:
In the main function, I created a GameState object using the GameState struct. Then it simply loops over the update and draw functions with a slight pause between loops to sync up with the game's frame rate.