Yew Yee Perng - 32205481

Report for frogger FIT2102 Assignment 1

Frogger is a classic retro arcade game that was popular back in the early 2000s. The game I created is based on the original frogger. Controls used to move the frog are WASD keys to go Up Left Down Right respectively. The game ends when either the player lands the frog in a designated target area or when the frog dies of collision.

Summary of the workings of the code and highlight of interesting parts

To begin with a summary of my whole code, I begin with initializing the types and classes. Then, I created a function called gameClock where it takes care of the time for updating the new states every interval time. Then, several functions such as KeyObservables, Leftmove, Rightmove, Upmove and Downmove that takes care of the keyboard movements. Then, I headed of to declare the attributes in each type such as Rect, Circle and state. A function called createRect is then created used to create the rectangles which would be used in the games as cars and logs in the game. Then, a function called handleCollisions is used to handle the collisions between the frog and other objects such as cars, logs and the river. Then, a function createfrog is used to create the frog, our main character of the game. Then, a function initial state is used to document the starting state of the game where when the game ends and restarts, it goes back to the initial state which is the start of the game. Then, moveBody is a function that considers the movement of the obstacles like logs and cars. On the other hand, reducestate is used to detect the movement of the frog. For every move the frog advances up, the player is awarded 10 points. After that, a function named Updateview is used to update the svg scene. Finally, subscription is a function that is the main game stream.

There were several interesting parts in the code. The first interesting part I would like to highlight is handleColisions. The main idea of this function is to check if there exist any collisions between the frog and the obstacles. First, we find the vertical and horizontal distances between the center of the frog and the center of the obstacles. If the distance is greater than half of the frog or half of the obstacle, then they have no chance of colliding due to far distances. However, if the distance is lesser than half of the frog or half of the obstacle, they are absolutely confirmed to be colliding. Besides that, it tests for collision at the obstacles corner. It tests by checking if the line from the center of the obstacles to any obstacles corner are colliding with the radius of the circle if the circle appears to be on the line. Pythagoras formula is used to compare the distance between the frog and the obstacle centers. If the frog collides with any obstacles, the game is over. A link of referenced page is included as a internal documentation to show the inspiration source of the code. Another special part of this code is when the function named RectCircleCollisions which was in handleCollisions was called to check if the collisions of each obstacles happened by using a filter to check if the collision happened. If it returns any number above 0, the collision is true since any number above 0 in Boolean is true. The second interesting part I would like to highlight is moveBody. The idea behind this function is to check whether the obstacles get out of bound. If the obstacles get out of bound, it spawns again on the other side of the canvas depending on the direction of the obstacle. I find these interesting as it was one of the rather harder implementations in this assignment and I had a struggle while implementing until I got the concept behind it.

A high-level overview of your design decisions and justification

The game created is based on the original frogger game. Design decisions mainly follows the original frogger game with the concepts of it.

Explain how the code follows FRP style

Observable streams are created throughout the game either from user events or automation from the code. Asynchronous behaviours can be handled following FRP style. The code follows FRP style by following a declarative type of programming style.

How state is managed throughout the game while maintaining purity

The state maintains purity by using built in functions such as map and filter. For example, map creates a whole new list or array based on the input without modifying the original input array or list.

Describe the usage of Observable beyond simple input

Observables streams are created from keyboard events for simple inputs. For events beyond simple input, it could be based on the progress of the game while the user plays which correlates with the game clock from tick where there could be automated observable streams created through automated due to the interval time of the game clock.