Frogger Report

Prepared by: Ong Jing Wei
Student ID: 32909764
10 September 2022

Table of Contents

Body of Contents

This is a report on the classic arcade game, Frogger which uses Functional Reactive Programming techniques (FRP) which allows us to monitor asynchronous operations like user interface events in streams, specifically the Observable/Observer pattern. This program imported rxjs library since it has a large collection of operators which is useful when handling asynchronous tasks.

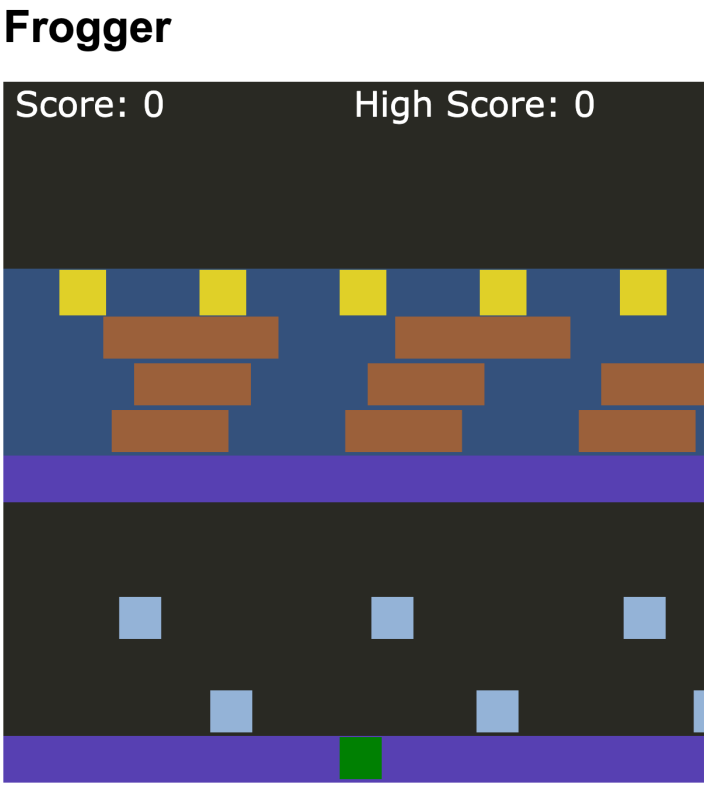

Diagram 3.1

Diagram 3.1 above is the output of the frogger code. In this game, we are required to move the frog (green rectangle) using keyboard to the distinct target area (yellow box) to win the game. The frog must cross through the road and avoid colliding with the obstacle (blue boxes) and stand on the log (brown rectangle) as frog will die when contact with the river. Scores are recorded once the frog are able to reach the target area and the high score records the overall highest attempt score. This program uses observable to listen for the user input and it manipulates the state of all the objects and update it to the view.

Overall, this program has 3 main important parts to make it function which is the gameClock, reduceState, initialState, and updateView. First, the purpose of having the initialState is to initialize all the elements for example the polygons inside the diagram 3.1 and also the game state which we need to keep track of such as the win or lose status and also the game time at the beginning of the game. Next, the gameClock has the time interval function which keep increasing throughout the whole game makes all the elements moveable by increasing their coordinate with the timestep. The reduceState keep listening to the user event and make changes to the state of the objects and returns the new game state, where the new game state is passes to the updateView since all the changes made is then need to update to the SVG by

getting the element's attribute and update it. The most important function in this program is gameClock. It has a Tick class which acts as a simple class to store the discrete timesteps and to pass it into the reduce state. We merge the gameClock to the stream, so that all the objects are able to move by just updating the coordinates, such as if we wish to make an object move from left to right, we get the x-axis of the object and add a certain value to it. Hence for every tick, the functions will keep triggered and the objects coordinates will change, and it seems like the object is moving.

By using Functional Reactive Programming and observables, it helps us to maintain the code purity by separating the pure data and the data transformation from the impure svg updates. The observable operator scan is implemented to observe the state transformation inside the stream. Instead of directly altering the state values, this scan function applies the reduceState function onto the initialState and creates a new output state object with new changes and update it the SVG. Besides using scan operator, we also used Readonly for the state to avoid making changes to the value to maintain purity. The only impure function is the updateView where all the possible transformation of the state is pass in to show it to the SVG. The merge operator is applied to combine all the asynchronous stream into one stream as to carry out all the event at the same time. Side effects are avoided as much as possible to prevent unpredictable function depends on the state of the system.

We also used keyObservable to listen for the keyboard event by filtering the desired key and apply function to each key. In this, we use classes instead of interface to store the movement of the frog as we are able to use the function 'instanceof' to access to the class inside the reduceState. Whenever the user clicks on the keyboard, it filters on the key and place the value inside the class. The value which is then able to access by the other function using the class. In addition to the reduceState, it keeps listen to the keyboard event, and it also constantly calling the tick function where this function handles the collision event among the objects. If there are no collisions, then the objects coordinates are keep changing for every time interval as to make it move. All those changes are made and is updated by returning the state.