**FIT2102 Assignment 1 Report**

**Lim Zheng Haur**

**32023952**

## Summary

This game uses the Model View Controller Architecture that divides the program into 3 elements which as the name implies is the Model, View, and Controller. This allows us to separate between pure and impure programming as SVG updates are impure while we could maintain as much purity with our data transformation as possible. To relate it to the game, the Model is the state of the game, the View is the SVG renderings and the Controller is the streams and the scan function with its other supporting functions such as reduceState, etc.

## Design Decisions

I have declared each type as Readonly<> and hence, I could ensure that my functions that transform these data are pure and returns a new instance rather than mutating the data impurely. To achieve the effect of turtles floating and diving into the river, we set a Boolean attribute to the turtle type so that if the Boolean is not satisfied, we ignore any collisions and the frog is considered not standing on any floating objects, hence dying. The Boolean is then flipped every 5000ms by tracking the game time in the game state. I believe this is a simple yet elegant solution. Moving onto the restarting game functionality, to complete this functionality, my reduceState function will return a copy of the input state of the game if the game is over, hence the game will seem to be stuck in the same state until a 'r' key is detected. Then reduceState will return a new state which is identical to the initialState for the scan except for the new updated highscore array.

## Functional Reactive Programming Style

Functional Reactive Programming style processes streams with pure functions and allows us to avoid loops. I have applied this principle throughout the code of the game. For example, with the tick function, rather than mutating the argument state input, it creates a new state and returns it with updated values. In addition, such as in updateState function, rather than using loop which is not functional programming, I use forEach function which applies the input function to every element in the array and returns a new array. Although updating the attributes of the SVG elements is impure, we try to maintain as much purity as possible by only using these impure functions in updateState function. Ensuring that each functions created is pure and using these functions to mutate the state of the game, we adhere to the Functional Reactive Programming Style throughout the game.

## State Management

An initial state is initialized at the start of the program. For every event of the stream, the reduceState function is called to handle the transformation of the state according to the event passed through the stream. There are a few additional functions such as tick, and handle collisions

which are used to manage the state of the game. To maintain purity, all these functions take in a state and returns a new state rather than mutating the argument passed in, enforcing FRP principle.

**Observables**

The usage of observables is the core of Functional Reactive Programming which allows us to capture the events in streams. In this game, I used fromEvent to capture the keydown events of the keyboard to capture the user input for the frog movements and a specific key to restart the game. However, it is used more than just to capture the events of the user, we used an interval stream in order to capture the passage of time in the game, using it to tick each of our objects and handle collisions between objects in the game.