Student ID: 30720591
Name: Rebecca Nga
FIT 2102 Assignment 1 Report

Minimum requirements report (summary)

Assignment requirements	**Notes and Justification**
Frog can move forwards, backwards, left and right using the keyboard	Key observable is used to let the frog move upwards, downwards, left and right with keyboard keys (W, A, S, D)
Multiple rows of objects appear and move across the screen	List of objects created: 1. Car objects (3 cars) 2. Plank objects (7 planks) 3. Log objects (3 logs) 4. Other objects (snake, snail, stone, bee)
Objects move at different speeds and directions (left-right)	Objects moving: 1. Cars a. Car (yellow) - right to left b. Car2 (red) - left to right c. Car3 (blue) - left to right 2. Other objects: a. Snail (brown square) - right to left b. Bee (yellow square) - left to right c. Snake (yellow rect) - upwards to the ground The objects move using an animate function and timer function, to set attributes of the direction of the objects and the interval of the object
Correct collision behaviour, at least one ground section and one river section	Ground (green long rect) - when frog lands on the ground, nothing happens, dies when colliding with car object in ground River (blue long rect) - when frog lands on the river, the frog dies, but when frog lands on a plank on the river, the frog doesn't die and can move on.
The game ends when the Frog dies	When the frog collides with objects (cars, bee, snail, stone, snake, river), the frog dies
Indicate score for the player	Score is incremented by 1 when the frog lands on the log (target area)
Player score by landing the Frog in a distinct target area	Score is incremented when the frog lands on any of the target area

Introduction

The purpose of this assignment is to create a classic arcade game using Functional Reactive Programming (FRP) techniques. My program was implemented in Typescript and used RxJS observable items to handle object behaviours in the game. This assignment was done referencing FRP Asteroids written by Tim Dwyer (https://tgdwyer.github.io/asteroids/) and some RxJS functions within observableexamples.ts from Week 5's tutorial, whereas the handling collision function is referenced from 2D collision detection - Game development | MDN.

Report

The first part of my assignment was to write a checklist based on the requirements provided and work through them slowly according to the order, all my code is implemented within the main.ts file and minimal code is written in the index.html file containing the instructions to the game, the scoreboard for the player and current status of the frog of the game displayed for the player. At the beginning of the assignment, only a few state transitions were initialised, those transitions were created as classes and later as more functionalities were needed, more state transitions were added. A game clock is created to track the movement of the frog in the entire game. Then, types of different objects and elements are declared to maintain functional purity before the state of the game is implemented. This will ensure that the types of the object will not be mutable later on in the code. The game state is implemented mainly using Readonly to help maintain functional purity. Using Readonly would help the game state be strictly immutable, this is also to prevent the object types from being changed illegally later on during the game code.

Firstly, the river(water) and the ground are created with createRiver() and createGround() function and they are designed first so that the frog object would have to go through both of these parts before reaching the target area, they are also the biggest objects within the game and they have objects placed on top of them. Cars (3 cars) are created through the createCar() function, to be placed on the ground and move at different speeds and directions, after rounds of testing, if the user decides to move the frog too fast, a collision between the car and frog would occur. Other objects such as stone, snail, bee, and snake are created within the createObjects() function, I've grouped them here to organise various types of objects. Next, the planks (7 planks in total) are created with the createPlank() function on the river to act as stepping stones for the frog to cross the river and reach the target. The logs are then created through the createLog() function as the target area for the frogs to land on.

To move the objects, animate functions are created below each of the create functions for the specific object and the timer for the animation of the objects. This keeps my code organised and if there are errors, all the code is grouped up and can be found easily. The movement of these objects is done within the code and cannot be manipulated with keys or arrows, only the movement for the main frog object (the main character) will be done using key observables that was similar to week 5's tutorial, and the frog will be able to move within the game. Initially, arrow keys were implemented, however after adding instructions to my game, the

arrow keys would let the page move downwards, resulting in a bad gameplay experience, so the use of the “w”, “a”, “s”, “d” keys were implemented instead.

I have tried my best as possible to follow the FRP style by using the game state, maintaining the game state, using the states to update characters, creating a function to handle collisions between objects with the states, and implementing an initial state to keep track of each initialisation of game state’s, a function to reduce state and update view where reduce state holds the states of transitions that are updated in these functions and update view function keeps tracks of all the changes made to the state, and a tick function to track the time and progress of the game. Overall, trying to make functions as pure as possible.

Conclusion

Overall, the assignment was challenging to keep the code pure and follow the FRP style, there were many trials before the final submission, but the assignment requirement was followed and completed at a minimum scale. There were attempts to continue to implement requirements and improve the overall game code, such as restarting the game and including more enemies for the frog with different functionalities and features. However, only the minimum requirements have been met due to time constraints.