FIT2102 Assignment 1 Report
Name: Sayyidina Shaquille Malcolm
StudentID: 32578393

- Introduction

  This is an implementation of the Frogger game created in 1981 by Konami. This implementation is written in Typescript using Functional Reactive Programming. This program uses the RxJS library to implement reactive programming. By doing it this way, the program becomes as pure and functional as possible

- Game Design
  1. The game accepts the arrow keys to move left, right, up, and down. The game also accepts the space key to reset the game, if the player decides to reset the game while in the middle of the game.
  2. The main destination of the game is to go to the end of the road while avoiding cars and using logs to not drown in the river. The frog also can't get swept by the logs outside of the bounds or it will die. The frog also needs to land at the designated areas (colored as red) and can't land at the areas with spikes or it will die.
  3. Everytime the player reaches the destination, the moving object on screen will be faster. Hence, making the game more challenging as the player moves on
  4. The game doesn't have an end, so players can always try to get the highest score possible.
  5. The score is incremented by 10 every time the frog moves upward.
  6. The player also can reset the game in the middle while the game is running to reset everything to the initial position.
  7. Road Area
     1. There are 3 lanes in the road area
     2. In each lane there are cars moving either left or right
     3. If the frog collides with a car it will result in a game over
  8. Safezone Area
     The middle lane and the starting lane are the only 2 safe zones where the frog can move freely without having to worry about the game being over.
  9. River Area
     1. There are 3 lanes in the river area
     2. In each lane there are logs moving either left or right
     3. If the frog touches the river (or the blue part of that area) it will result in a game over
     4. The frog can use the logs as stepping stones to safely cross the river and not drown.
     5. if the frogs is on top of a log, the frog can ride the log hence, moving the frog the same speed as the log
     6. If the frog keeps riding the log until it's out of bounds, it will result in a game over

FIT2102 Assignment 1 Report
Name: Sayyidina Shaquille Malcolm
StudentID: 32578393

- Observables
    1. Observable Stream
       The entire game is dependent on this one stream i called the mainStream. Everything from the discrete time step to the key observables that listens for the user's input is flowing through the mainStream stream. We merge all of those to a single stream. And then we use scan to apply a reducer function with an initial value of initialState (this is where the entire state of the video game starts).

    2. reduceState
       `reduceState` takes a `State` and an `Event` and returns a new `State`. If the Event is a Direction object it will call a function called *moveFrog* where it will move the current frog with the data taken from the Event parameter.

       If the Event is a Tick object,it will call a function called *tick* where it will change the positions of the objects while simultaneously checking for collision and any other game-winning/game-losing conditions and it will return a modified State to be used in the next iteration of scan.

       If the Event is a Reset object which in this case is when the player inputs the spacebar key. It will overwrite the current 'State" back to the initialState where when returned, everything in the game will be in its set initial position.

    3. handleCollisions
       This function checks the frog whether it has collided with any game-losing object such as cars or checks if the frog has satisfied any game-winning conditions such as arriving at the destination rectangle. This function is also responsible for increasing the speeds of the objects when the frog has arrived at one of the destinations

    4. updateView
       This is where the code changes the value of everything that's displayed on screen. Because it's changing mutable global variables, this is the only impure function in the code. With only 1 impure function, this prevents from having any side effects on the code.