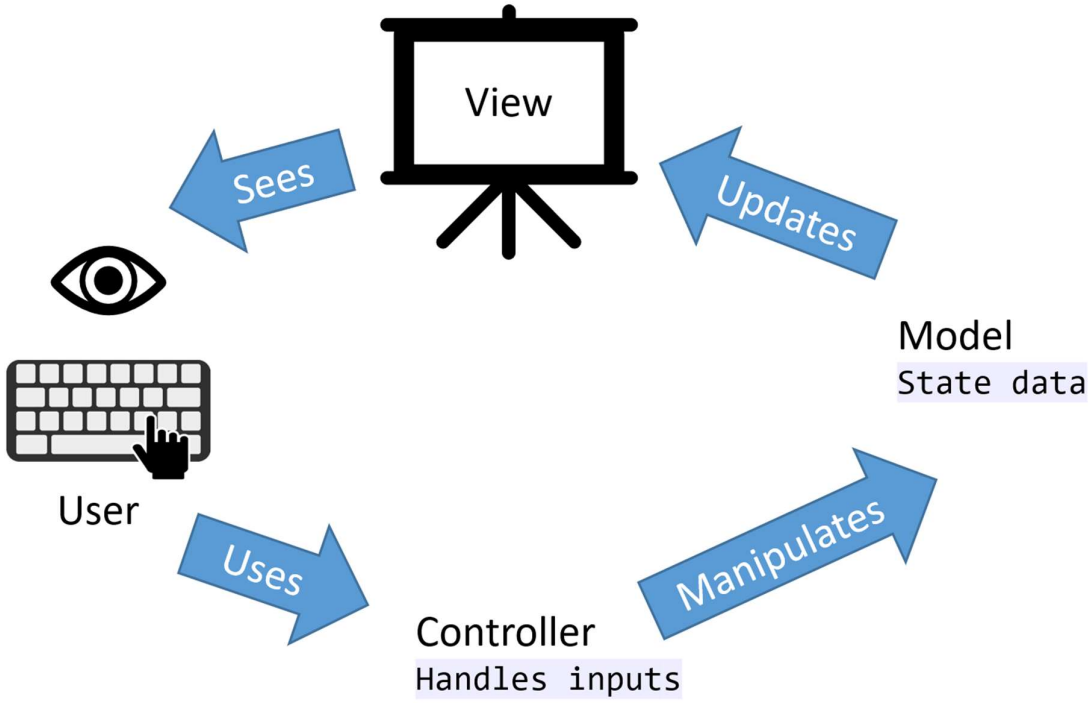
FIT2102 Assignment Report

Name: Jin En Tan

ID: 31336574

Introduction:

The implementation of my Frogger game uses the Functional Reactive Programming. These allows me to capture asynchronous user interface event in streams. Furthermore, it allows me to avoid using deeply nested for loops when coding the game. The architecture that I used in implementing the game is the Model-View-Controller (MVC) architecture.



The diagram above is taken from Tim's asteroid reading material and it clearly shows the MVC architecture.

Model:

State Initialisation:

Type State is defined in my program in order to keep track of current game state as well as to store the initial game state. Initial state of the game is hard coded in order to initialise all

the objects and variables needed in game. Type State is Readonly to prevent modification to the attributes inside. Each state is independent to each other and any changes of current game state will result in returning a newly created game state so as to prevent side effect. This allows state to change throughout the game while maintaining the purity of code.

View:

updateView function is the only impure function under the FRP design. I use this function in order to reflect the status update of states onto the screen and this is the reason that it has side effect because any output to screen is considered as side effect. Under the MVC architecture and FRP design, I managed to contain all the side effects inside this function to have minimal side effect in my program and that it is separate from other parts of code.

Controller:

This is where the observable is created along with instantiation of a few classes in order to perform type checking using “instanceof” in reduceState function in order to detect keyboard inputs from user as well as the game clock stream.

Game Design:

Objects in Game:

I decided to use rectangles with different colours to represent all the objects for the game. Objects in game are frog, car, plank, destination, river, safe zone (which is goal of the game).

Player/Frog:

In my implementation of game, player can control the frog to move left, right, up and down by pressing “A”, ”D”, ”W”,”S” respectively and the game can be restart by pressing the spacebar. I implemented the frog to be able to move along the plank automatically when standing on it to make it more challenging for the player.

Car:

This is the obstacles that can cause frog to die, collision behaviour is implemented so that frog will die and game over immediately when it collides with car. Certain rows of car move to right whereas certain rows of car move to left. Each row the car moves with different speed to fulfil the requirement of the game.

Plank:

This is the rectangle that frog needs in order to cross river to reach destination. I implemented so that some rows of planks move from right to left whereas some move left to right. It is similar to car during implementation just that it does not have the same collision behaviour as car. I also implemented so that the frog when landing on plank, will move automatically with the plank to make it more challenging.

River:
River is a rectangle on canvas that frog will dies instantly if it lands on it.

Safe Zone:
Safe zone is a rectangle on canvas where there's no moving object crossing that row.

Collision behaviour:
I implemented simple checking to check for collision between frog and car using just the vector, width and height. There's also additional collision behaviour implemented to check for collision between frog and river, frog and plank as well as frog and destination. Several functions are created in order to achieve those purposes. A frogRespawn function is created to move the frog back to the game starting position once it reaches the destination.

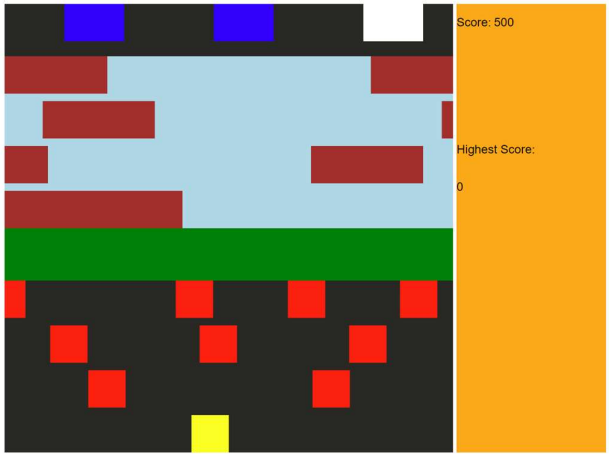
Lose condition:
The losing conditions are when frog collides with car and frog collides with river. The highest score that the player has achieved will be recorded so that player can keep playing to beat the previous highest score.

Winning condition:
There's no winning condition for this game, player earns score by successfully moving the frog into the destination block. Once all 3 destination blocks are filled, the game will restart keeping the current score until game over.

Functional Reactive Programming:
I used rxjs library for observable implementation and typescript type annotation to make sure that variables are immutable to prevent side effects. The use of imperative loop is avoided using FRP and I did not use any "random" in my program to avoid any side effect. Curry function is created as many as possible as it can delay the running of functions until all the required parameter is given. Curry function is very useful in this program as it allows me to create partial application/function such as when I create the rectangles for different object.

Conclusion:
This report is just a brief overview of my game implementation. Most of my codes are ported or wrote based on Tim's asteroid code. Refer to the source code for further details of functions written. Picture below is the screenshot that proves that my game functions as expected, after reaching 3 distinct destinations it can restart automatically keeping existing scores.

Frogger



Controls:

Move Left	A
Move Right	D
Move Up	W
Move Down	S
Restart Game	Space

Rules:
Avoid red block and light blue area to avoid Game Over!
Green area is safe to rest
Brown block is safe to step on
Move towards white block to earn Score :)
Move towards same white block will earn 100 score but GameOver!!!