Report of code design

Name:Neevashinie Paramasivan
ID: 32267037

The code could basically execute the minimum requirement of the gameplay where the frog could move in all directions using the keyboard arrow keys. A restart state which restarts the entire level when the game is over by pressing the space key.I divided the rows 5 that indicated the road and another 5 indicating the river. A middle lane was left between the road and river indicating a safe zone. All the cars and logs operate in different speeds and directions. There are collisions implemented between car and frog and river and frog. If the collision returns true the game will end. There is a section that shows the control keys, score and highscore. The control key indicates the frog's movement while the score increments each time the frog moves and the score will increase by 50 when it hits the small box at the end of the target. At the end of the map there are 4 small boxes that indicate the checkpoints in which when the frog hits the small box it will turn green and the game restarts where the frog goes back to the beginning position till it hits the other checkpoint.

Design and justification

I designed this entire game highly inspired by tim's asteroid code which uses multiple classes to simplify the access to certain classes. I also used const to implement certain functions such as createCarRow and createLogRow. I also created multiple constants to call certain aspects of the obstacles,frog,canvas and checkpoint that can be accessed at any point of the code. Lastly, I implemented an updateView function that adds all the elements into the Canvas and shows their physical form.Unfortunately some of my designs have some problems, mainly my frog, does not appear above the log rather it appears below it, nevertheless the game can be played as usual just by keeping track of the frog's movement below the log.. I also did not match the frog's speed together with the log, to make it seem like it is moving together. Thus, we have to move the frog manually to match the logs' movements This makes the game harder as we have to catch up with the logs movement and not collide with the water. I also designed a checkpoint feature which is supposed to turn each of the checkpoints from red to green if the frog has landed on it and it should return the frog back to its original position to move to the next checkpoint. This is to make the game more interesting for the fact that the player had to finish all the checkpoints to win the game. Unfortunately, some of the functionalities of the checkpoints are not working as intended.

Explain how code follows FRP style

All these code uses higher order functions to derive the complex structure. For example, the collision code has multiple const of collision methods embedded. Each

of this collision code uses a high order function to make comparison and execute the collision. For example, the frog_log_collision checks whether the coordinates of the log collides with the coordinates of the frog. This method is actually used to derive the water and frog collision, in which when the frog collides with a log and river is it safe but when it is collided with only the water it will end the game. This collision is then called in the tick which handles all moving objects and executes it in the reduced state. I also followed the key stream code to implement the keyboard keys for the frog's movement and for restart of the game.

How is state is managed throughout and how it maintains purity

I declared a state type which holds all the states that are going to be used in the game. Then I used const to create an initial state which stores all the initial form of the states for example, the initial position of the frog, cars, log and initial scores. I used a tick method to manage all the moving states namely the car and log which ensure that both of these states move. Lastly, I have a reduced state which combines and executes all the states in the canvas using the subscription method. These states also maintain purity by only returning the same input with no side effects to it. For example, the reduce state function uses the states passed to it and returns the states without any changes implemented to it.

Usage of observables

Observables are highly used when creating the movement of the objects and appending the objects to the canvas. For example the updateView function reads from the svg and appends the item created within to the svg.The updateView function holds all the frog, car,log and checkpoint object being appended to the canvas. The updateView also holds 2 additional features which are the gameover and the checkpointAppear function. The gameOver unsubscribes the game and displays a large text stating game over. Meanwhile the checkpointAppear function changes the checkpoint colour from red to green whenever the frog reaches a certain checkpoint. Unfortunately, the checkpoint is currently not working how it is supposed to be, thus the functionality is faulty. The updateView also holds the view for the score and highscore. The score is calculated whenever the frog moves in the correct position and the highscore is added together with the score. Both of these are displayed outside of the canvas.