FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

General Summary

All the code in the game has been organized in the following way (from top to bottom): classes, custom typings, and functions, with the most important functions all at the bottom of the code.

The game mainly runs on the ***tick*** function which updates the game every 10 milliseconds. It is responsible for checking conditions which may result in game over or a reset of the game state. The tick function passes the state of the game into the ***collisionHandler*** function which checks for collisions and returns a new state of the game after collision checking. The state is then passed into the ***reduceState*** function which transduces the state and subscribes to the ***updateView*** function which is responsible for updating the view of everything on the canvas.

All the functions in the file are pure in that they return a new immutable object and have no side effects besides the ***updateView*** function as it is responsible for updating the view of the canvas through ***index.html***.

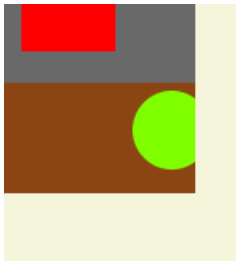
FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

Design Decisions

Wrapping

The first design decision I made was to have the Frog be able to wrap around the map to allow the Player freedom of movement. The ***torusWrap*** function in the ***Vector*** class allows wrapping around the map seamlessly for **continuous movement**. However, the Frog is designed to move around in **discrete movement**, hence using the original ***torusWrap*** function would let the Frog move itself to a position such that it would be half inside and half outside the canvas.



Thus, I decided to create another function called ***frogTorusWrap*** that checks if the Frog's next move will be outside of the canvas and wraps it around the map if so.

On the same topic, another design decision I made was to have the Frog not be able to jump across to the area where the targets are located if it is a yellow area which acts as a wall of sorts.

FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

I implemented this by using the same concepts in implementing ***frogTorusWrap*** and had ***frogTorusWrap*** additionally check if the x-coordinate of the Frog was aligned with the range of the x-coordinates of any of the targets before allowing it to jump across.

Managing State

The way I managed state throughout the game while maintaining purity was by ensuring every function always returned a new instance of the state based on the current state passed into the function using the spread operator. I also made sure to declare everything as const as well as declaring Readonly where possible and not using any imperative code in any functions to prevent mutations.

Collision Handling

My take on handling collisions was by having the function ***collisionHandler*** constantly check the difference between the positions of the Frog and every object in the game. This is done by wrapping the state of the game at the end of the ***tick*** function inside the ***collisionHandler*** function before having the ***tick*** function return

the new state. To elaborate further, the ***collisionHandler*** function has another function ***bodiesCollided*** compare the result of subtracting the position of the Frog from an object in the game to the radius of the circle representing the Frog and half the width of a rectangle which represents every other object in the game with different sizes. This works for colliding with land objects such as cars, buses and snakes as well as the targets. However, it is a different case for objects in the river, crocodiles, turtles and planks.

I implemented the whole river section by first having the Frog collide with the river if the Frog was above a certain y-coordinate and tracking that fact using a boolean property of the Frog called ***inRiver***. Then, I implemented standing on river objects by checking the opposite of colliding with crocodiles, turtles and planks using the '!' operator.

Similar to using ***inRiver***, I had the boolean properties of the Frog ***onCroc***, ***onTurtle***, and ***onLog*** each represent whether the Frog was standing/colliding with any of them. These are important as they are used to match the velocity of the Frog with the velocity of the objects to emulate the Frog moving together with the object in the River.

FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

Target Filling

One design decision I made was allowing the Player to refill “filled” targets, however they would not receive points for this. A problem I encountered while implementing this was since there are three targets to fill in the game, if the Player filled the same target or two of the same target and another once, the game would see it as the level beaten. I fixed this by making the targets have a ***filled*** boolean property and having the ***tick*** and ***collisionHandler*** functions check in the state whether all three distinct targets were filled.

Another design decision I made was to have the Frogs remain inside the targets after they are filled. I implemented this by having the ***updateView*** function create a new Frog object and mapping its position to the Player’s Frog’s position before resetting the game. On reset, the Frog filling the target remains as it is not part of the state. It is instead removed by concating its ***id*** to an array called ***removables*** which removes all the elements of the ids inside the array on restart or when the level is beaten.

FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

Snake Bite

Besides the car/plank object collisions, the snake is the second distinct object with an interaction/behavior. When the Player collides with the snake, the Player will turn into a snake and move in accordance with the rest of the snakes forever. This interaction is

in a sense an infinite loop as the Player will not be able to do anything anymore besides restarting the game.

I implemented this feature by giving the Frog a snakeBite boolean property and having the ***updateView*** function check the property before turning the Frog into a snake. The movement was implemented in a way similar to how the movement for the Frog on river objects is implemented.

Crocodile Timer

The third and final distinct object with an interaction/behavior that I have decided on is the crocodile. If the Frog stands on any crocodile in the game for more than 2.5 seconds, the Frog will get eaten by the crocodile it is standing on. This is implemented by having a timer ***timeOnCroc*** increment every tick the Frog is on the crocodile and resetting it otherwise.

FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

Extension: Double Jump Power Up

One additional feature I have in the game is the double jump power up. On pickup, it allows the Frog to jump twice the distance. The powerup spawns randomly on the map every level and provides the Frog with this ability until it either dies or completes that level.

I implemented the random spawning by using the ***RNG*** class which was inspired by the one shown in Dr. Dwyer's PiApproximationFRPSolution video. The ***RNG*** class is a stateless class which maintains purity and emulates the use of Math.random() here which is impure. The ***RNG*** class is deterministic as the random number generated will always be the same for a given state, hence the ***next()*** function inside the class is used to create a new instance of the ***RNG*** class in which the ***int()*** function of the previous ***RNG*** object will be passed in to constantly create new random numbers.

To allow the ***RNG*** class to constantly call ***next()***, a ***rng*** property inside the state is used. The state calls ***next()*** inside the ***tick*** function every tick which in turn creates a new random number every tick.

Now that generating random numbers has been achieved, the state ***rng*** can now be used to generate random numbers to determine the position of the powerup inside the canvas.

FIT2102 Assignment 1 Report

Name: Ooi Yu Zhang
Student ID: 32713339

The way I implemented double jumping was by simply checking the Frog had the ***doubleJump*** ability and if so double the number of steps it jumped. I also had to create a new version of ***torusWrap*** called ***doubleJumpTorusWrap*** to accommodate for the changes and to ensure the Frog still couldn't jump into the yellow wall or out of the canvas.