FIT2102 Assignment 1 Report

There will be four main functions in my code. One for starting the game, ending the game, keyboard controls as well as collision. I created a main observable called myObservable which will control all the events throughout the whole program. This observable is also responsible for starting and ending the game. To account for closure principle, most operations inside the function will only access the variables present within the enclosing scope. An example was the ball_reset function which accesses the p1 and p2 defined within the function.

For the design choices of my code, each segment will be control by an observable and the observable will be subscribed one at a time to perform operations such as setting the states of the ball, paddle and collision. There are also observables that will consistently update the content of HTML text.

To support the concept of parametric polymorphism, I created a generic method called setAttr which takes in a string or a number value to update the current attribute of the specified object.

The getAttr method was also created which is derived from getAttribute method to allow for short and concise form.

Starting the Game

The first section is the start_game function, where myObservable will control the flow of the game by keeping tabs of player's score. The moveball function was created to direct the ball towards the player's side at the start of every turn and controls the movement speed of the ball. This section is also where the AI paddle is controlled and moves based on the ball's direction.

Collision

In the collision section, I created multiple parameters for the collision function to cater for possible scenarios for when the ball hits the paddle. The ball can hit up to four sides of the paddle. One for the top quarter, top middle quarter, bottom middle quarter and bottom quarter. Edge quarters will always increase the speed of the ball faster than the middle quarters. To handle the state of the ball, for when collision happens, I

declared a ball_state which contains the attributes x, y and speed. At each stream of data, ball_state will constantly update the current x, y direction as well as the current speed (y-velocity) of the ball. This was done in adherence to functional reactive programming, as side effects was minimized, and no mutable variables was used.

KeyBoard

The next section is the keyboard function where I defined a state interface to capture the current state of the player 1's paddle. To change the current state into a new state, the function updateState deep copies the old state and returns a transform state. In order to prevent side effects, all attributes are set to read only therefore making it immutable. The updateState function also satisfies the referential transparency concept as the "y" property of the paddle was correctly substituted without affecting much of the program. The current valid keys are "W", "S", DownArrowKey and UpArrow Key.

Scoring System

The penultimate section is the ball_reset function where it activates whenever the player or the AI scores a point. In this case, two states were used which was the ball_state and the score_state. The score_state keeps tracks and updates the current players' scores and the ball_state was used to set the ball back to the middle.

Ending the Game

The last section is the end_game where the winner is announced based on which player's score reaches 7 first. When the game ends, the player will have the option to restart the game by clicking on the screen. This was implemented using the click event and was assigned to an observable. To keep the stream of data clean, all objects from the previous state of game were removed. Side effects were contained appropriately by limiting the mutations to the subscribe call of the click observable.

No mutable variables was used throughout the program and only states and constants were used to maintain function purity.