***FIT2102 Programming Paradigm Assignment 1 Report***

This report is to generally discuss and highlight my design decisions throughout this assignment.

**[General design thinking]**

Functional Programming is all about functions, so I tried to functionalise as much as possible in this assignment, for example creating a function if possible when there exists duplicate code. Based on that, the entire code is trying to follow the MVC architecture with the FRP style that captures asynchronous actions into a merged observable stream, and then manipulate the model based on the conditions with subscription to update the view once the manipulation completes. The functions inside are guaranteed to be pure by setting everything as const or “Readonly” if possible, and always returning a new object instead of mutating the values, especially for the <state> and <body>. The only impure function which is indicated in the code is the Math.random() function that is used to set up the initial position of the alien bullets so that the game is different each time. The way I tend to manage the state throughout the game is by setting up a state type that consists of everything needed for managing the game state just like in the asteroids example code, and then map the state into different functions or a pipe of functions depending on the event type to achieve the corresponding manipulations, with each function returning a new state to maintain the pure property.

**[Summary of the code]**

Basically all the elements in the key event streams are converted into corresponding classes in order to process them, these streams are merged with the tick stream (implemented using interval(10)) which is used to update the state every 10ms into one to manipulate the state based on each action in the stream. All the other classes except Tick() like Move() and Fire() are for handling the corresponding events in a systematic manner, the Tick() class is where the most of code contributes to. Once Tick() is observed in the stream every 10ms, it will enter the first layer which is for updating the countdowns and the positions of the ship and bullets, as well as filtering all the expired objects based on a given specific time, into an array called “exit” to inform the “updateView” function to remove the elements. In which the countdowns are just variables (however it won’t vary as we always return a new one instead), that are used to achieve the different velocities of the movable objects like aliens rather than moving every 10ms, without having multiple interval streams. We will then return the updated state as a new state to a function that is used to handle different kinds of collisions. After collisions are handled, it will arrive at the third layer, which is a function that acts like a check point to see whether all aliens are shot so that to avoid moving “ghost” aliens, instead it will be led to another function that restarts a higher-level game, otherwise it will enter the function that handles the alien moves. We want to move the aliens down when the left or right border is reached or end the game if the aliens reach the ground, the function that handles the alien movement does take these into account. One interesting point is that, since the aliens are created and maintained in sequence and we reverse the aliens’ moving direction at the time they are moved down as what the game should look like, therefore in terms of moving them discretely (i.e., one by one), to find the next alien to move down, the responsible function simply detects it by looking for the first one that has a different moving direction than the previous ones.

Upon these, the updated state that comes out from this pipe of functions will be given to a function to update the view.

**[Design decisions]**

1. For moving the ship: I decided to make the ship be able to move around the svg instead of implementing the hard walls since I think it should be interesting to implement something untraditional.
2. I used a set of overlapping circles to form a shield, as it gives a very similar look as the one in the demo video, and it achieves the same purpose without complicating the code.
3. I used the remainder and the quotient with the index to calculate the initial position for each of the aliens as well as for the shields, although it may not be the best idea, but it assigns a unique position for each of the aliens without having unnecessary functions.
4. I created a pipe of functions (A set of functions such that one passes its result to the next one) for each Tick(), as there are a lot of manipulation required to maintain the appropriate state, so in such a way they won’t get too messy, and it is also corresponding to the functional programming style.
5. I also tried to keep all the crucial data in the CONSTANTS object, so that they are easy to manage while making sure they are immutable.
6. Slowing down the fire speed of the ship and speeding up the fire from aliens for each level up requires the player to have a better hit-rate while in a more challenging state.
7. Constant countdown variables described above are stored in the state, they are implemented to avoid multiple unnecessary interval streams while achieving different velocities for the objects.
8. Bullets from the ship can also deteriorate the shields just like in the demo video, this makes the game more challenging and interesting.

**[Future work/Potential Improvements]**

The code can certainly be extended to come up with a more interesting game, such as increasing the speed of the aliens per level up by changing the countdowns, giving the player extra lives while increasing the difficulty of the game by having a new element in the state, etc. The code can also be improved to avoid the use of Math.random() that has the impure property to make the code entirely pure.