Student ID: 31171184 | Student Name: Tan Jie Yen

FIT2102 ASSIGNMENT 1 REPORT

I started this assignment by restructuring the given asteroids project by Mr. Time Dwyer. Firstly, I modified the spaceinvaders.html and style.css file based using previously self-learned knowledge. I created a division in html file to display score, high score, life and level and defined the styles for these elements in a separate CSS file. To organise my project neatly, I put all the typescript files into a src file. To start coding, I made use of the asteroid05 file as a template, looked through it and tried to understand the code to a general idea of where I would begin my alterations.

Files:
In src file, there are 4 files: spaceinvaders.ts, helper.ts, utils.ts and models.ts.
The Helper.ts file contains functions which I can import and use within my spaceinvaders.ts file to create an array, define object images based on their view type and more.
The Utils.ts file contains utility functions cited from asteroid05 and other functions I created, for example one of them simply helps me to generate a random integer
The Models.ts file contains all the class type and interface.
By structuring my project in this way, I've been able to separate code that I plan to reuse time and time again – this keeps improves the readability of my project and improves its efficiency.

Mouse Event:
For the mouse event where user clicks the screen to start the game, I use the fromEvent function to create an Observable stream. In this stream, the menu button is hidden and all the svg contents are cleared, these are my initial settings. After calling the spaceinvaders function to start the game, I unsubscribe from this event. A menu button is created using division in the html file and I also make it visible before the game starts. Once the button is clicked by using MouseEvent, the button will be hidden again.

Keyboard Event:
For all the Keyboard events, I use the rxjs fromEvent function to create an Observable stream called keyObservable. KeyObservable is a reusable function as the pattern of creating each observable is the same. In the stream, I then use pipe() to compose the operators which are filter and map to transform the stream. The required key will be filtered out and at the end of this pipe, map() is used to return the result.

Game Settings (Score, High Score, Alien Speed, Life, Level):
The settings of the game will be stored here. The score will be incremented based on the length of the collided aliens list. Calculating the score by referring to this list helped me avoid using mutable variables. The high score will be updated as soon as the user loses, and if their score has surpassed the previous high score. The score, life and level of the game will always reset to 0 when a game ends. The game will proceed to the next level when all the aliens are killed. Furthermore, the alien speed increases each time the player proceeds to the next level.

Creating Objects:
I referred to the createCircle function in asteroid05 and made some modifications when creating aliens, bullets, shields and lives. To ensure that the aliens and shields start at the position required, a flat list that merges the x-axis and y-axis of the objects is created for both aliens and bullets by using the flatMap function cited from asteroid05.

States:
All the possible transformation of states will be encapsulated in a function. By using merge, I was able to merge different streams and scan over every state. The stream will then subscribe to call the updateView function and also check if the player is proceeding to the next level.

Student ID: 31171184 | Student Name: Tan Jie Yen

FIT2102 ASSIGNMENT 1 REPORT

Update View:
This is an impure function cited from asteroid05. In this function, I updated the view of the svg scene based on the position of each object. The collided objects that cease to exist get removed here. The game settings are updated here as well.

Alien Moving Action:
Moving the aliens was accomplished by using an Observable stream at an interval of 1 second. The alien will move 10px every time the interval observable emits an event (every 1000 milliseconds).

By using scan, I was able to get the occurrences every time the stream was emitted. Once the aliens have moved horizontally by 10px 4 times, they then move downwards. The direction of alien movements are decided by looking at the last digit of their occurrences. 1 – 4 implies they move right and 6 – 9 implies they move left. For every 5 occurrences, the aliens will move downwards.

In order for the player to proceed to the next level, the time taken in interval Observable needs to be changed. Therefore, I came up with a formula which is (1000 / alien speed). The time taken in interval observable will decrease when proceeding to a new level due to the increase in alien speed. I then merged the following stream with a stream that will reset each alien's velocity back to zero to stop them from moving and to achieve a **discrete** movement effect.

Alien Shooting Action:
An Observable stream was created to randomly choose an alien to fire a bullet every second. This randomness was achieved based on each alien's respective id, in the range of the size of the remaining aliens. In order to avoid an impure implementation of Math.random, I made use of Mr. Tim Dwyer's RNG code. By using this class, the output generated from every same input will always be the same, keeping the code to be pure.

Collision Handling:
The collision handling of objects in my game is the same as the colliding bullets in the asteroid05. For all collisions, I use the cut function from asteroid05 to get only the remaining objects. The collided objects that need to be removed will be stored in an exit list. I then implemented a norepeat function that uses the some() function to ensure bodies with same id do not appear more than once. This prevents an error from occurring when removing body in updateView function.

Add Life Action:
This is an extra feature I have created that extends the player's gameplay. An Observable stream was created and is emitted every 10 – 14 seconds. In this stream, pipe is used to compose the map operators that will create the add life action to transform the stream. If the spaceship collides with a heart, the user's life will be increased by 1.

Shield to protect Spaceship:
A shield is simply many "smaller" shields that were combined together to form a "big" shield. There are a total of 4 big shields in this game. Both the collision between a small shield and an alien bullet or spaceship bullet will cause the small shield to cease to exist. I found it difficult to mimic the "eroding shield" effect. This implementation is the same way I've handled other collisions, which was a lot easier than the eroding shields displayed in the provided YouTube video.

Body Movement:
There is an edge wall wrapping the width of the canvas. To prevent the spaceship from always avoiding an aliens' shots, the spaceship movement does not exceed the width of the canvas. Since a

shot bullet's expiry time is not the same for each row of aliens, the bullets will go exceeding the height of canvas.

Game Over:

- Life becomes 0
    - Spaceship collided with bullets from aliens
    - Spaceship collided with aliens
    - Alien reached bottom of the canvas

As soon as the spaceship collides with aliens' bullets or aliens, the life of the player will be decreased by 1. The game will end once the player's life hits 0. If such a scenario does not take place, then to prevent the aliens from moving out of the canvas, the game ends and the player loses once an alien reaches a position lower than that of the player's ship.