FIT2102 Programming Paradigms Assignment 1: Space Invaders Report

Due Date: 10th September 2021 (Friday 11:55 PM)

The assignment was to create the classic game Space Invaders using RxJS Observables with a pure functional style which have minimal side effects.

My implementation draws heavily from Prof. Tim Dwyer's implementation of Functional Reactive Programming (FRP) example of Asteroids: https://tgdwyer.github.io/asteroids/

Table of Contents

How the game works

FRP allows us to capture asynchronous actions from events such as user interactions and intervals in streams.

Similar to Asteroids, Space Invaders is implemented with time-based intervals known as ticks, which corresponds to user interaction. (see usage on line 516)

To maintain pure observable streams, a state that stores parameters is used.

This allows the creation of pure functions which takes the input state object instead of altering the state in-place, thus creating a new output state object with any necessary changes required. (see line 73). To adhere to FRP style, pure functions are crucial as they keep side effects to a minimum. The only impure function I've used was the updateView() function.

Observables are lazy, in order for us to make anything happen, we have to subscribe to it. The subscribe function will trigger an independent execution for that given observable.

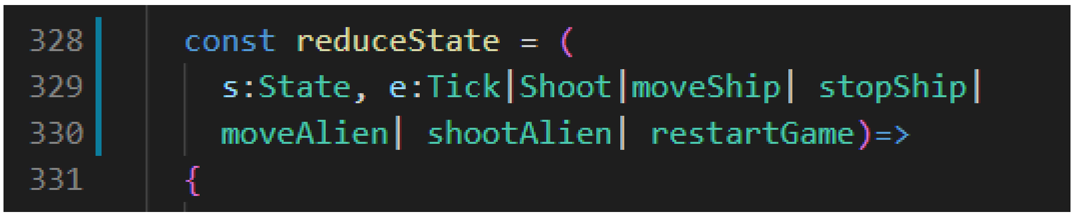
I implemented the game using an interval-based stream which incorporates vectors which helps represent the movement/velocity of different components of the game. By keeping track of position and velocity, we can easily manipulate the bodies in the game by adjusting values using the vector class.

Events and State

In the main observable, every event input is considered as an instruction which tells the game how to transition to the next state.

To follow the rules of FRP, a new state is constructed and then passed into the reduceState function which encapsulates all possible transformations of the state.

To ensure immutability, an interface was defined for state with readonly members.



```
const reduceState = (
  s:State, e:Tick|Shoot|moveShip| stopShip|
  moveAlien| shootAlien| restartGame)=>
{
```

The **reduceState** function/(const variable) handles most of the logic of the game.

All possible transformations of the state are encapsulated in reduceState.

reduceState is passed into the scan (an Observable operator which allows us to capture the state transformation inside the stream, using a pure function to transform the state) which was seen in the first screenshot on page 1.

A **Tick** event revolves around the time in the system. Examples include moving objects (aliens, ship, bullets) based on their velocities in their previous state and checks for different triggers such as collisions with other bodies, which in turn triggers a change in the state.

More events such as Shoot, moveShip, stopShip, moveAlien, shootAlien are pretty much self-explanatory. The events play a role as a trigger or indicator which tells the reduceState function what new state to create.

The flow of the game:

The state updates the view using updateView() which then looks for user input such as Shoot, move, which in turns calls the reduceState() function to manipulate the state.

Design Decisions and Justifications

Movement of Aliens

The aliens move horizontally upon creation, this was the implementation similar to Asteroids where each object was given an initial velocity, which dictates how the object moves.

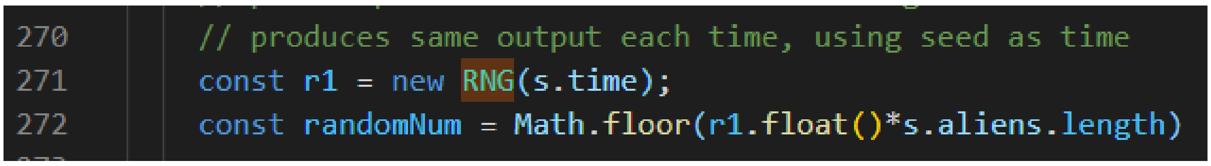
(line 126) function spawnAliens dictate the velocity/speed of alien's movement

Another design decision to highlight is that collision of the alien's body and ship would mean that it was unnecessary to check if the aliens reached the bottom border of the canvas but did not touch the ship and would still result in a loss. However, in my implementation, as the alien moves horizontally throughout the whole canvas and eventually reach the row position where the ship is at, the aliens would not miss the ship, thus it results in the same as reaching the bottom border and resulting in a loss for the player (Game Over).

Alien Bullets

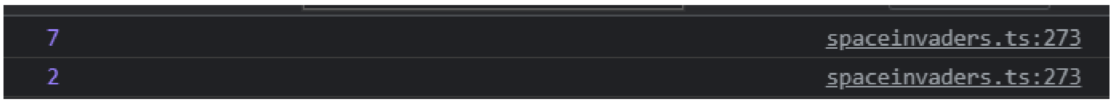
Only a single random alien will fire a single bullet and its bullets will not collide with other aliens below it. The bullets will only collide with shields or the players ship.

To achieve a pure pseudo-random number generator function, the use of RNG from Tim's Video was used:
https://www.youtube.com/watch?v=RD9v9XHA4x4&ab_channel=TimDwyer



```
    // produces same output each time, using seed as time
    const r1 = new RNG(s.time);
    const randomNum = Math.floor(r1.float()*s.aliens.length)
```

The seed takes the state's time in the system. We can check that the function is pure when we console.log(randomNum) and check the console in the html page. The alien which fires will always be the same according to the time, so refreshing the page we would notice it is always the same alien firing.

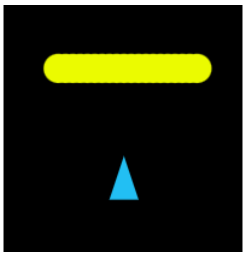


```
7                                                   spaceinvaders.ts:273
2                                                   spaceinvaders.ts:273
```

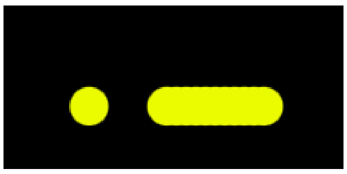
Alien id: 7 will always fire first, followed by alien id: 2 when we start the game.

Shields

The shields are just a bunch of circles clumped together to make it look like layer of protection with a smooth look.



Upon collision with the aliens bullet, the shield will disintegrate according to the bullet's radius/impact thus leaving gaps. This looks better than creating a bunch of rectangles and removing them completely upon impact. The shields would also vanish upon collision with the aliens body.



When moving onto the next level, the shields damage is maintained in the transition from the previous state to the next state. This makes the progression appear more realistic and harder for players, rather than refreshing the shields for each level.

The reason for there being only a single layer of the shield above the players ship instead of multiple layers is because it already demonstrates the knowledge of handling collisions and maintaining the state, adding more layers above the first layer of shields would not complicate things but only make the game easier for players.

Ship Movement

I thought it was better to not allow the ship to move past the border of the canvas and reappear on the opposite side of the canvas and instead modify the torusWrap function adapted from Tim's Asteroid code to not allow any body to get past the borders. (line 171).

Restarting the Game

The decision I came up with was to allow users to restart the game at any point in time was because it felt like a waste to only allow users to restart when the game is over (either victory or defeat). Thus, the control panel also shows player the instruction to restart by pressing "Enter". This was handled by returning the initial state of the game, thus adhering to pure function convention.

The game restarts instantly without player input when they are defeated (ship was hit by bullet or collided with alien body) because that would incentivise players to win the game. However, when the player does win, the option to restart is displayed as well.

"Future Work/Potential Improvements"

For now, I only demonstrated that progression in the game is possible where levels and the score are carried over by maintaining properties of previous state through events such as defeating all aliens, but a power up was never shown visually in game.

Possible powerups:

- implement the ship having more than one bullet at a time
- aliens move much faster in higher levels as well rather than just periodically after bouncing off the borders.
- More than one random alien shoots
- Shields have health instead of disintegrating after one collision with alien bullet