

FIT2102 Assignment 1 Report

Name: Darren Yee Jer Shien

Student ID: 31237223

Introduction

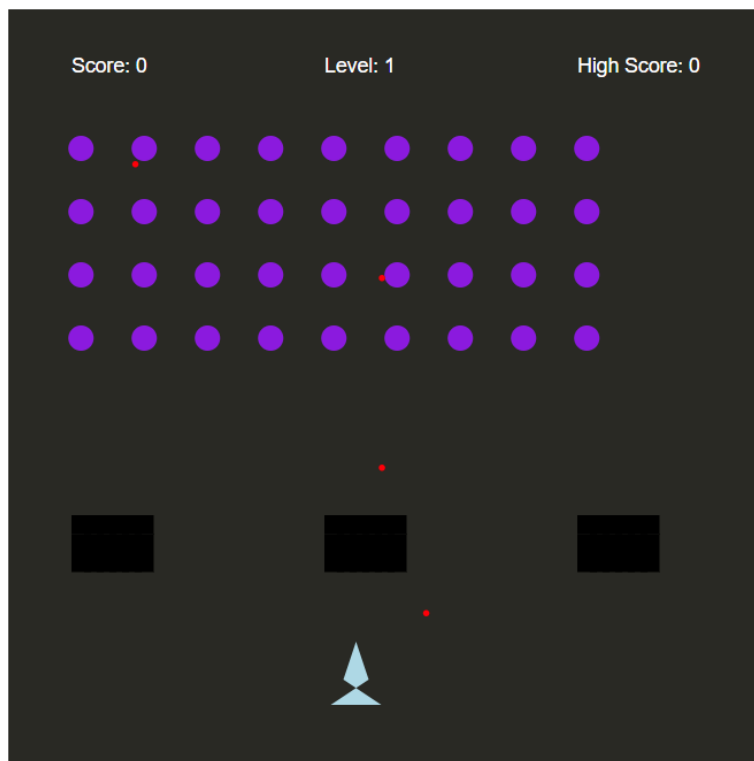
The aim of this assignment is to create a clone of the space invaders game using Rx.js. In this report, I will describe my implementation in steps.

Constants

To keep the game more streamlined and easier to amend when testing, I have decided to use constants extensively for important numbers of data such as the number of aliens, the number of shields etc. This allows me to change and play around with balancing the game without needing to change numbers one by one and risk breaking the code.

Basic Gameplay

In the html, controls will be shown which will help the user to be able to know what functions each button is for. The button pressed will also be highlighted (adapted and modified from Asteroids FRP) to show the current button pressed to remind the user again what the buttons do. The player will be able to move left and right in the game. During the initial iterations, the ships can go up and down and will also fly continuously when button pressed (i.e., when the right key is pressed, the ship will fly right until stopped.). However, I decided against it and changed it to its current behavior since it resembles more of the classic space invaders game that we are trying to recreate whereas the initial behavior is more like the Galaga (another classic arcade game). Other than movement, the player can shoot bullets at the enemy while the enemy will shoot back at the player.



Each time the enemy is shot, score will be added and once the ship is shot, the game will be automatically restarted. The decision to implement is this way and not the way described in FRP Asteroids (unsubscribing the entire stream) was since unsubscribing does not allow the user to save

FIT2102 Assignment 1 Report

their highs – score throughout while trying to beat the game since it is not possible to resubscribe the stream and that the only way to do it is to without refreshing the page is to reset the state into an initial state. The code for this was done in a manner that would go with the principles of FRP since instead of causing side effects since the initial state will be returned instead of the making changes to the data of the current state. Unsubscribing the stream only happens at the end of the game in this implementation when the user beats the game (there will be 5 levels which will be discussed more in depth later) which will display a message that allows the user to press W to refresh the page and start over from level 1 (since there is no point recording the high – score anymore after the player beats the game).

Bullets

In my implementation of the game, there are two different types of bullets (bullets that come from the enemy and bullets that come from the player). There are major differences in behavior between these two bullets since both (the direction they travel, their hitbox etc.) which is why I have decided to implement them separately. This way, it allows be to be able to differentiate within the game between bullets so that the aliens (enemy) will not be hit by friendly fire. This was done by using different FRP methods such as map and filter to make sure that there are no side effects.

Levels

There are a total of 5 levels in the game. During each level, the aliens will move progressively faster, and their firing rates will also increase. There is a level attribute within the state which is used to calculate how much the increment will be in the movement of the aliens and the firing rate. I accomplished this by incrementing the y by the amount of level that the player is currently at which means that the higher level the player is, the further down the aliens will move each time. Although there was a choice of keeping the game progressing infinitely (no level cap), I decided against since the levels past a certain point would be impossible anyways. After each level, the game will be returned to the initial state with the only change being the level attribute.

Collision

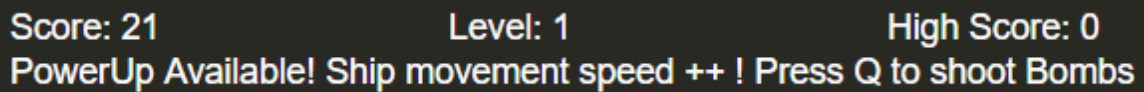
Collisions are handled in three different ways here (ship collision, body collision (aliens / shields), as well as bomb collision (powerup that will be discussed later). The decision to separate them was since each of them must have different hit boxes to allow for more accurate registration of damage. The collision is handled using different FRP methods such as map, filter etc. to make sure that side effects are well contained instead of having side effects. This part was partially influenced by the FRP asteroids notes along with methods such as cut that are given which allows us to remove the unwanted items (expired bullet for example) out from our existing list.

```
const handleCollision = (s: State) => {
  const bodiesCollided = ([a,b]:[Body,Body]) => (Math.abs(a.x-b.x) < 10 ) && (Math.abs(a.y-b.y) < 10 ), //hitbox for normal body collision (aliens, shield)
  bombsCollided = ([a,b]:[Body,Body]) => (Math.abs(a.x-b.x) < 70 ) && (Math.abs(a.y-b.y) < 70 ), //hitbox for the AOE (large hitbox) bomb
  shipCollided = ([a,b]:[Body,Body]) => (Math.abs(a.x-b.x) < 15 ) && ((a.y-b.y) == 50 || (a.y-b.y) == -30), // hitbox for ship
  alienToShip = s.aliens.filter(r => shipCollided([s.ship,r])).length > 0, //when aliens hit the ship
  bulletsToShip = s.bullets.filter(r => shipCollided([s.ship,r])).length > 0, //when bullets hit the ship
  allBulletsAndAliens = flatMap(s.aliens, b => s.bullets.filter(r => r.viewType == 'bullet').map<[Body,Body]>(r => ([b,r])), //list containing all bullets and aliens
  allBombsAndAliens = flatMap(s.aliens, b => s.bomb.filter(r => r.viewType == 'bomb').map<[Body,Body]>(r => ([b,r])), //list containing all bombs and aliens
  allBulletsAndShield = flatMap(s.shield, b => s.bullets.filter(r => r.viewType == 'bullet' || r.viewType == 'bulletEnemy').map<[Body,Body]>(r => ([b,r])), //list containing all bullets
  allAliensAndShield = flatMap(s.shield, b => s.aliens.filter(r => r.viewType == 'aliens').map<[Body,Body]>(r => ([b,r])), //list containing all aliens and shields
  collidedBombsAndAliens = allBombsAndAliens.filter(bombsCollided), //list containing all bombs and aliens that have collided
  collidedAliensAndShield = allAliensAndShield.filter(bodiesCollided), //list containing all aliens and shields that have collided
  collidedBulletsAndAliens = allBulletsAndAliens.filter(bodiesCollided), //list containing all bullets and aliens that have collided
  collidedBulletsAndShield = allBulletsAndShield.filter(bodiesCollided), //list containing all bullets and shield collided
  collidedBullets = collidedBulletsAndAliens.map(([_ ,bullet]) => bullet).concat(collidedBulletsAndShield.map(([_ ,bullet]) => bullet)), // all bullets that have collided with something
  collidedAliens = collidedBulletsAndAliens.map(([_ ,aliens,_]) => aliens).concat(collidedAliensAndShield.map(([_ ,aliens,_]) => aliens)).concat(collidedBombsAndAliens.map(([_ ,aliens,_]) => ali
  collidedShields = collidedBulletsAndShield.map(([_ ,shields,_]) => shields).concat(collidedAliensAndShield.map(([_ ,shields,_]) => shields)), // all shields that have collided with somethi
  collidedBombs = collidedBombsAndAliens.map(([_ ,bombs,_]) => bombs), //all bombs that have collided with something
```

FIT2102 Assignment 1 Report

PowerUps (extension)

Different from the base game, there are two different powerups that can be activated by the user that is unique to this implementation. First and foremost, after the player kills 20 of the aliens, they will gain the ability to shoot a bomb. The bomb looks identical to normal bullets (but larger) and has a large hitbox that will deal AOE (Area of Effect) damage to surrounding aliens, causing them to explode. It lacks the proper effects (explosion) which is something that I intended to add on however due to the time restriction, it was not possible. This allows the users to be able to counter the increase in speed and firing rate that the enemies will gain when there are only 10 of them left. The second powerup added is that after 20 enemies are killed, the player will experience a surge of energy and be able to move left right with higher speed. This is done by increasing the addition / subtraction in x.



Score: 21 Level: 1 High Score: 0
PowerUp Available! Ship movement speed ++ ! Press Q to shoot Bombs

State

As mentioned briefly in the previous paragraphs, the main way state is handled in this implementation is that instead of directly amending the values within the state, I have decided to make important attributes read-only (bullet list, alien list etc.). This allows me to follow the Functional Reactive Principles since I only use pure functions which does not have any side effect. In other words, I am simply creating a new state and returning it every time I update my state with the new data to ensure that there are no side effects. Furthermore, the reduce state function within my implementation also goes well with FRP Principles since we are only amending the data within the scope of a function instead of outside.

Citations:

FRP Asteroids :

<https://tgdwyer.github.io/asteroids/>