FIT2102 Assignment 1

Space Invaders Game using Functional Programming with TypeScript and RxJS Observable

Name- Sadeeptha Bandara
ID -30769140

This report is regarding the Space Invaders game implemented for this assignment and aims to summarize the workings of the code as well as to describe the design decisions taken and how they are justified.

Code Architecture

The code architecture is inspired by the asteroids example provided as part of the course notes. It involves a MVC (Model-View-Controller) architecture.

MVC is particularly fitting for systems that have to respond to asynchronous events. It enables you to decouple state management from the view elements leading to a more cohesive system with well-defined responsibilities.

The flow of the game in this implementation is first listening to a stream of asynchronous events which are processed to an appropriate form and then used to manipulate the state of the game. This is done using decoupled, self-contained pure functions while state management is done using immutable state objects. Finally, the state is used to display the change to the user.

The design decision to follow this architecture has provided a number of advantages, which justifies the decision taken.

- Decoupled state management from the view makes the code more extensible as it is not dependent on it beyond clearly defined input streams
- It also makes the code more coherent and well structured, leading to a more declarative style of coding
- The decoupling lends itself well to a functional style of programming, allowing to process the state with pure, referentially transparent functions
- Different components of the system are interchangeable with a different implementation, making the code resilient to changes over time, leading to a more maintainable system.

Feature Implementation and the design decisions involved

1. Handling Asynchronous events

The main use case for handling asynchronous events in the game is for listening to user interface events. This is handled by using RxJS observables. Using RxJS provides a number of benefits as opposed to following a more traditional event listener model

- It allows to linearize the flow of control, by considering asynchronous events as streams which can be composed, filtered, combined, with each other allowing to manipulate the data as we wish.

Handling complex user interface events with event listeners requires multiple callbacks to be nested and would require them to placed in such a fashion that variables are appropriately scoped in order to perform the required operation. This leads to more messy and less declarative code, of which the flow of execution is not linear.

2. State management

The flow of the game as described above, pertains to the manipulation of state according to the input streams provided. However, it must be noted that the change of state is not done with a global mutable state, but rather with immutable state objects that are passed between different functions.

The functions themselves are pure and referentially transparent, containing no side effects. No mutable variables are used, processing is done with functions over imperative code blocks.

The game flow occurs with by using a 'game clock' an observable interval that fires every 10milliseconds. The use of observable is very helpful in achieving this.

The primary reason for this decision is because it leads to a decoupled architecture, which is declarative in nature and provides the benefits mentioned above.

3. View

Encapsulates all code that involves performing side effects. It is decoupled from the rest of the system and takes in a state object in order to perform its task

In all of these proponents, the code is modularized in to functions for reusable code blocks, as well as for structuring the code base. It is written with type safe code which no implicit any types. Generic functions are identified where possible and bloated code was refactored.

This provides a number of benefits that justify these decisions

Notable feature implementations

1. Handling collisions

Collisions are handled by approximating the relevant bodies to be elliptical for the purpose of modelling the collision.

Fig 1: Approximated ellipse for the ship.

All of the assumed constants, including radii are declared at the top of the code.
Each of aliens, ships are considered to be a single ellipse, which a shield is actually modelled with three ellipses.

This approximation was shown to be effective and was sufficient for most cases

2. Only aliens that have no aliens below them fire bullets

This feature avoids potential alien casualties, due to their own bullets, and is done in a functional style by filtering out the aliens in a row and sorting them, and flattening the array

4. Shield deterioration

Shield deterioration is achieved by drawing black dots on the collided areas. However, not only are they just drawn, bullets will filter these collided areas, such that they are allowed to pass through.

5. Speed of aliens increase as the count of aliens decreases.

Achieved by scaling the speed vectors considering the speed of the aliens. In a similar fashion, the different aliens will have a different score associated with them.