Name: Darras Darras ID:34335277

Introduction:

This report aims to explain the programming design choices made while working on Assignment 1 of the course FIT2102. In this project, a “guitar hero” style rhythm game was created using Functional Reactive Paradigm, JavaScript Language and it leveraged the RxJS library for efficient handling and management of game state. While playing, players interact with notes represented by circles that move on a canvas, by pressing keys on their keyboard, to play the notes alongside the background music.

Code overview:

The game logic was executed by using a tick$ stream that emits when a certain time frame has passed, dictating the pace at which other functions run. The “State” type encapsulates all game related data, such as current score and notes to play, etc. Only specific streams were allowed to change the state, which made it easier to trace any issues back to their source. With each tick or user input, we update the state of the game in order to progress with the gameplay.

I decided to base my approach, when it comes to determining what notes are to be loaded, on the current time which meant I had to make sure that the game’s internal clock ran precisely and properly. In addition, I had to make sure that while parsing the notes from the csv file, the notes being input matched the clocks precision level in order to perform comparisons correctly.

Functional Reactive Programming:

In this project, data was stored as constants to ensure that they could not be mutated in any situation, instead they had to be reassigned in order to undergo updates, for example how the game’s state was handled, which enforced immutability. A stream was set up exclusively for accumulating states and managing which action can update it.

Another important aspect is how to handle side effects, such as rendering frames or playing music. In this case they were grouped up in a single subscription and allowed to execute there only, which helps ensure other functions in the project are pure (Refer to visual representations section). As to why we want to ensure that, it is because having impure functions while using observable streams can result in unexpected output such as a note being played multiple times instead of just once.

Using observables doesn't just span over dealing with inputs, it also allows the code to execute asynchronously and perform real time decision making based on the current state. These two features are what I capitalized on in order to schedule notes.

Functional code only executes when it needs to do so, this is called “lazy evaluation”, and it helps code run efficiently and smoothly by eliminating the overhead of having to calculate further than what is needed for the meantime, hence why it was used in the game.

Possible improvements:

The current game starts out by filtering for playable notes through the list of all user-playable notes, which introduces the need for extra processing at the start of the game and its effect decreases with time as played notes are regularly removed from the list as time passes. What I could have done to improve this issue is to introduce a way for the code to stop filtering after it encounters the first non-playable note. This is a solution that takes advantage of the fact that the notes in the csv file are already ordered due to the chronological nature of music.

Name: Darras Darras ID:34335277

VISUAL REPRESENTATIONS:

The diagram below is made to describe the flow of data through different streams in my project.