FIT 2102 Assignment 1 Report

In this report, I will focus on how I designed the Guitar Hero program and mostly explain the code (e.g. why do we use it, what is it for). My Guitar Hero program consists of 5 main files: **main.ts, type.ts, state.ts, and util.ts**.

State Management (state.ts)

The state module comprises all the state-related logic and is built at the center of game architecture. The state is also a modular and immutable object, which means that when we update the game state, rather than changing the reference of where our existing game state. This is very important in terms of keeping functional programming (FP) pure, that functions should always return the same value given the same input and it does not have any other side effects.

The state object consists of properties like score, gameEnd, circles, notes, etc. Each of these properties keeps track of the state as the program runs. For example, the score will keep track of how many points the user has gained when playing the game.

In the updating of state in the program, we have action classes included in the program, such as Tick, RemoveCircle, UpdateCircleState, and GamEnd classes, these classes all implement the Action class. Every class that implements the Action class will consist of an apply method, that it will take in the current state and return to the new state. The updating classes are very important in the program, as they ensure that our data is up to date as the program runs.

Type Definition (type.ts)

The type.ts included all the types, interfaces, and constants used in the program. All the types defined in the program will be immutable. For example, the type state, is defined as a Readonly indicating that the value inside the types is immutable which is the core principle of Functional programming (FP), this ensures that the state can only be changed through the updating section which is all the classes mentioned above in the state management section.

There are types that are defined as an array such as NoteObject and CircleObject, it will be declared as a ReadonlyArray in this case. This means that it is an array that cannot be changed, for example, if we try to add another element into the array it will be declined.

Utility (util.ts)

The util.ts is a Utility module, it is bundled with functions for creating SVGs as well as processing notes and handling user interactions.

Using the principles of Functional Reactive Programming (FRP) in concert with RxJS Observables to control asynchronous events, which is critical for a rhythm-based game. Functions like createUserPlayNote$ and createBackGroundNotes$ transform Note arrays in observable streams which can then be consumed by the rest of our operators so that we can manage logic around time-base events. This has the benefit of abstracting away some complexity involving simultaneous game events, such as showing notes and user Key presses.

Main File (main.ts)

The main. When run, this typescript file starts the game's core loop for handling user input (e.g., keydown events), updating your game state, and rendering it. It uses RxJS to build reactive streams for user inputs. The code combines both streams (e.g., key presses and note timings), to handle state transitions as well as visual updates, in real-time so the game is responsive and interactive. Its design is very modular, making it scalable and easy to maintain.

Conclusion

In conclusion, the principle of FRP did apply to it while creating the program. It ensures that all the state in the program is immutable and the use of RXJS for handling asynchronous events.