FIT2012 Report

Introduction:

This report provides an overview of the implementation of a Tetris game using Functional Reactive Programming (FRP) principles. The code leverages RxJS, a reactive programming library, to create a dynamic and interactive game. The report outlines the code's structure, design decisions, and highlights the application of FRP and Observables.

Code Overview:

The provided code implements a Tetris game with various game mechanics, user input handling, state management, and rendering. Key features include tetromino movement, collision detection, and game over conditions. The rotation system used here is the original rotation system

Design Decisions and Justification:

Modularity: The code is organized into sections, enhancing readability and maintainability. Functions like moveTetrominoLeft, placeTetromino, and render functions such as renderGrid, renderBlock are designed to perform one specific tasks, making the codebase more manageable and easier to comprehend.

Immutable State: State is managed using the State object, which is immutable. This approach ensures the purity of functions and eliminates side effects, resulting in more predictable behavior and easier debugging.

Observable Streams: FRP principles are realized through Observables. User input is captured using fromEvent, combined using operators like merge, and processed with the scan operator. This design enables a reactive flow of data, enhancing code clarity and structure.

FRP Style and Observable Usage:

Reactive State Updates: The gameLoop$ observable updates the game state over time using the scan operator and accumulate it into the initial state. The usage of withLatestFrom combines user input with the current state, allowing for responsive and interactive gameplay.

Dynamic Rendering: The render function leverages Observables to update the SVG canvas dynamically. Blocks representing tetrominos and the game grid are added or removed based on the state changes, adhering to FRP principles.

Observable Beyond Input Handling:

Collision Handling: The collision detection mechanism utilizes Observables. The code checks for collisions and updates the game state reactively, enabling the game to respond to user input and collision events seamlessly.

Game Over Logic: The game over condition is detected using Observables which is the gameEnd element of the game state which are emitted by the gameloop$ observable when the game runs. When the game ends, the "game over" message is displayed by updating the SVG canvas with the game over element.

Issues

In the gameloop$, and impure declaration of nextState has been used using the let keyword. One suggestion to make this pure is to create separate nextState for each of the conditions and then return the tick(nextState) inside the condition

The use of math.random is not pure. This can be fixed using random number generator Observable

Advanced Task:

A comprehensive test file have been generated which test that all main components of the game work as expected

Conclusion:

In conclusion, the Tetris game implementation exemplifies the principles of Functional Reactive Programming using RxJS. The code's design decisions, modularity, and usage of Observables for state management, user input handling, and dynamic rendering contribute to a functional and interactive gaming experience. The application of FRP facilitates cleaner code organization, improved responsiveness, and enhanced maintainability. This implementation showcases the powerful synergy between FRP and game development, resulting in a functional and enjoyable gaming experience.