FIT2102 PROGRAMMING PARADIGMS (A1 REPORT)

Nikhita Peswani

31361552

I. Introduction

This document showcases a Tetris game created using Functional Reactive Programming (FRP) ideas. We use FRP principles in our Tetris game, and we use the RxJS library to manage asynchronous events and data streams.

II. Code Summary

- The game is primarily made up of functional elements such as handling block movements, rotations, collision detection, and row clearing that describe game logic and adhere to the immutability principles.
- RxJS is used to record input events as observables, such as keyboard key presses. As a result, we are able to develop declarative event handling and slickly incorporate user input into the gameplay.
- Throughout the game, a State object, which represents the game state, is maintained. By applying actions (such as MoveBlock, Tick, or Rotate) to the current state and returning a new state with modified properties, state transitions are accomplished. This pattern that state changes do not have side effects.
- Blocks and other game elements are rendered in the game using SVG graphics. Based on the current state of the game, the render function updates the SVG canvas.

III. Design Decisions and Justifications

- Adapted a functional programming approach to ensure clean, maintainable code. This involved modeling the game as a series of state transformations, enhancing readability and testability.
- Utilized observables beyond input handling, controlling the game loop and timing. This separation of game logic from frame rate concerns encapsulates game time as a stream of ticks, aligning with FRP principles.
- Divided the code into separate files, one for constants and types, one for game logic, one for handling web displays, and a main file that coordinates all of these elements. The modularization improves the readability and organisation of the code.
- Utilized the observable pattern by employing RxJS Observables to capture and respond to user keyboard input events. Different Observables are created for distinct keys, enabling the program to react to user inputs like movement and rotation. These Observables are then combined and transformed using operators, resulting in a central tick$ Observable that represents the game's progression over time. The scan operator accumulates and updates the game state based on emitted events, while the subscribe method initiates the game loop and updates the game's visual display.
- In order to ensure that the game begins only after the web page has completely loaded, we must wait for the window.onload event. This strategy avoids the game starting too soon and ensures a consistent and predictable user experience.
- Used types and constants to define values that are consistent across the entire codebase. Through the centralization of configuration settings and the provision of precise type definitions, this practise improves the readability and maintainability of code.
- Employed the "Super Rotation System," technique. Tetrominoes can rotate around a central pivot point. After rotation, it determines new block positions to make sure that

tetrominos don't cross over with other blocks and remain within the grid's confines. This design decision offers players a simple and reliable rotation experience while allowing for different tetromino orientations.

IV. FRP Style and Observable Usage

- In FRP, a system is represented as an event- and data-flow flow. In our Tetris simulation, we model the game's actions and state changes as data flows, driven by observables like key$ and interval.
- To control the game loop (interval) and regulate game timing, we expanded the use of observables beyond input handling. This method separates game logic from concerns about frame rate by encapsulating game time as a stream of ticks.

V. State Management and Purity

- The state is immutable for the duration of the game. For each action taken on the state, a new state object is created. With this approach, state changes are prevented from having unintended consequences and code predictability is increased.
- Actions are developed as pure functions that take the current state as an input and output a new state. This functional purity facilitates testing and behaviour analysis of the game.

VI. Conclusion

To sum up, this Tetris game adheres to FRP principles and uses observables to regulate timing and event handling. The functional style, pure functions, and immutability all ensure the code's clarity and maintainability. By encapsulating game logic in a declarative manner, I created a fun game while adhering to FRP principles.