FIT 2102 Assignment 1 Report

Functional Reactive Programming

Classic Tetris Game

Heng Zi Ying | 33261865 | 2/9/2022

Abstract

In this report, we delve into the design and implementation of a classic Tetris game using SVG image, with a focus on how Functional Reactive Programming (FRP) ideas were used which is an approach to handling complex, asynchronous behaviours. This report explores how FRP techniques facilitate real-time game development, handle user interactions, and maintain a pure state, resulting in an engaging and responsive gaming experience.

Body of Contents

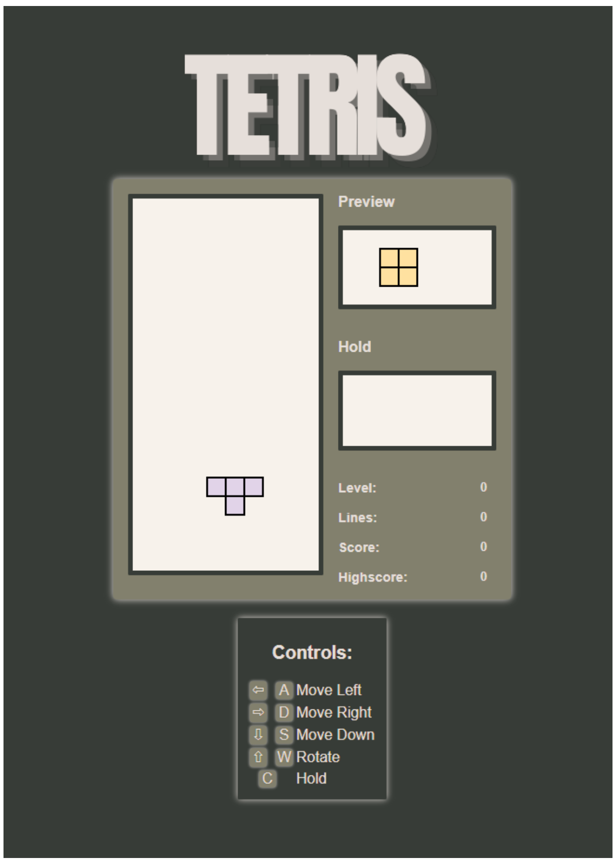

Diagram 1

The Diagram 1 shows the interface of the classic Tetris game. Classic Tetris game is a block matching puzzle game and to score in this game is to arrange the falling blocks that are in geometric shapes into complete lines. Thus, the score will increase when one or more complete line is found and level increases according to the total lines that the player completed through the game. Besides, when level up, the difficulty will also increase as the speed of the game will increase. The highest score will also be recorded through every round of game. In the controls, the Super Rotation System without wall kick is implemented for rotating the blocks and the hold function also implemented for the player to hold the block for the current round.

In the implementation, the game code is made up of fundamental parts that follow the Model-View-Controller (MVC) architectural pattern, such as state management, rendering, and event handling which can ensures code modularity and maintainability. Moreover, the heart of our game lies in the utilization of FRP principles to capture and process user input events as observables. For example, since this game required users to control the blocks when dropping (e.g. move left, move right, move down, hold blocks), so RxJS observables are utilized to create event streams that listen to the keyboard inputs that can transform into actionable game commands to update the state of the game and rendering it to the SVG.

Utilizing immutable state objects and adhering to the concepts of FRP, this game code maintains functional purity in state management. By setting the `State` objects to immutable using `Readonly<>`, which are regenerated with each modification, guarantees predictability and facilitates simple debugging, state management respects functional purity. The game state, encapsulated in the `State` object, is altered solely through actions, each implemented as classes adhering to the Action interface. These actions accept the current state as input and produce a new state, preserving the integrity of the previous one. Following pure functional principles, the `reduceState` function assumes a key role in computing new states without altering the ones that already exist. `InitialState` lays the groundwork and guarantees a fresh start for each game instance. This dedication to purity improves maintainability and reliability while also lining up perfectly with FRP's declarative approach, resulting in a stable and predictable Tetris experience. After computing the new states, the new state will be passed the `render` function to present it on the SVG.

This game relies heavily on observable, which goes beyond simple input processing to coordinate the game's essential features and state management. The observables for this game are mainly managing the heartbeat and block movement operations for this game. Amongst those observables that handling the keyboard events, one of the most important observables that act as a heartbeat of the game is `$tick`, it uses the `Tick` class to emits values at regular intervals, effectively acting as a game loop for the game. It is also used to increase the speed of the game. By merging various inputs into a unified observables stream, it can simplify event handling and maintains a consistent game rhythm. The `scan` operator and observable work together to maintain the game state in a practical, immutable way that improves reasoning and predictability. It allows for the implementation of intricate game elements like block rotations and collision detection while adhering to the concepts of FRP to produce a structured, modular, and maintainable approach to game creation.