FIT2102 Assignment 1 Report

Wong Yuan Yi 32845944

Code Overview / Summary

The game automatically starts after the browser loaded where it starts with level 1 and 0 points, where the objective of the game is to move left, right, or rotate Tetrominoes to stack them and clear as many rows as possible for a higher score. As more rows are cleared (100 points per row), the game increases speed per level. The game ends when the newly created tetromino spawns on top of an existing tetromino (meaning that the stack of blocks are at the top), also updating the highscore. Players are then only allowed to restart the game with the "T" Key (not allowed during the game).

The game follows Model-View-Controller (MVC) architecture, where the game starts out waiting for user input or a single tick instance via observables, then modifies a current state within the State class, and renders it for viewing using the render function. The controls are "W" "A" "S" "D" keys for tetromino movement and "T" key to restart the game. For each user inputs, Action classes are created from the observables and its apply function is called to update the state.

Design Decisions

- Each **Event class extends from Action interface** so "reduceState()" can be short and simple by just executing each classes' apply function, "instanceOf" can be avoided too.
- **Readonly Objects** and for all Objects and **ReadonlyArray Arrays** for all arrays are used to prevent manipulation.
- **1D array system** and 2D array system were considered to store static blocks, but 1D was ultimately used to reduce array indexing for a more FRP style.
- **Tetromino class** was used, because all tetrominoes are an array of 4 SingleBlock objects and other properties (type, rotationState and pivot).
- **RNG class** was implemented to create random blocks instead of using impure "Math.random()". The PC's current time in milliseconds is used for initial seed and subsequent seeds are obtained from previous hashed value.
- **Many Helper Functions** are implemented for updating State and rendering, because short functions can maintain clarity and can be reused. Example: "handleDown()", "handleClear()" and "updateRows()" are used in "apply()" functions of Tick and MoveY because both Event classes describe down movement of tetromino.
- **Super Rotation System** was implemented, as each block can be translated easily from the pivot with a function, and the implementation can be extended for wall kicks. Sources: https://tetris.fandom.com/wiki/SRS, https://www.youtube.com/watch?v=yIpk5TJ_uaI&ab_channel=TurboMakesGames

FRP and Observable Usage

The code is in FRP style by not mutating any objects or arrays and all functions used when handling the State is pure. Array indexing is also avoided as much as possible to prevent accidentally manipulating arrays (except when handling rotation, indexing is used to get rotation offsets). Instead, higher order functions that maintain purity, mainly “map()”, “filter()”, “reduce()” and others are used rather than imperative coding practices such as using loops. If Else statements are also kept to a minimum by using ternary operators instead.

Observables from the RxJS library are used to handle user input and game ticks purely. “fromEvent()” and “interval()” are the main observables used to instantiate Events that prompts to update State via “reduceState()”. “fromEvent()” creates observables from keyboard events and “interval()” creates observables at intervals to progress the game. Observables are also handled purely using pure RxJS functions, mainly “scan()”, “mergeWith()” and “subscribe()”

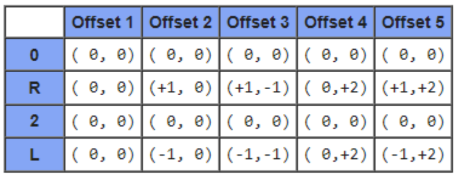
State Management and Purity

All Event classes that manage state extend from Action interface, where “apply()” function which manages state is mandatory. With State objects maintaining game state, only “apply()” and some additional functions (used to handle down movement and rotate) updates game state to reduce side effects. All of these are pure because they return new updated States where no manipulation is used. Other helper functions that update objects returns new updated Objects without manipulating them. Type hinting is also used in all functions to enforce types and reduce side effects.

Extra Game Feature: Wall Kick

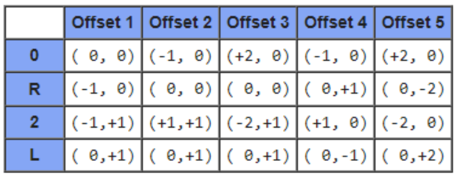
Wall kicks was an additional feature implemented, extending from the Super Rotation System, based on information from these sources: https://tetris.fandom.com/wiki/SRS, https://www.youtube.com/watch?v=yIpk5TJ_uaI&ab_channel=TurboMakesGames. With wall kicks, tetrominoes that are usually unable to rotate due to collisions can “kick” into a new suitable position by checking for suitable spots nearby. The idea of wall kicks is to first rotate the block normally by its pivot, then go through all offsets (Offset 1 – 5) and calculating an offset value based on the formula: (current rotation position offset - new rotation position offset) and adding the offset to the x and y values for each block in a tetromino. Example, when a “J” tetromino is going from rotation position 0 to 1, using Offset 2 would be (0,0) – (1,0) = (-1, 0). -1 would be added to x value while no changes to y value for all blocks in a tetromino. Once this is completed for all 5 offsets, each of them is tested for collision with other blocks, the floor, and the walls. If there is no collision, the offset that is available will be used as the new tetromino’s location. This method was implemented because it is very consistent with unintended side effects because the offset values were based on documentation from the official website.

J, L, S, T, Z Tetromino Offset Data



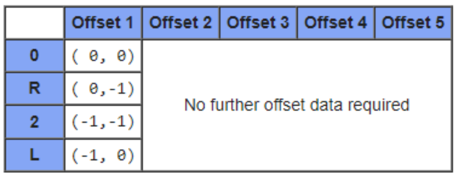
	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5
0	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
R	(0, 0)	(+1, 0)	(+1,-1)	(0,+2)	(+1,+2)
2	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
L	(0, 0)	(-1, 0)	(-1,-1)	(0,+2)	(-1,+2)

I Tetromino Offset Data



	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5
0	(0, 0)	(-1, 0)	(+2, 0)	(-1, 0)	(+2, 0)
R	(-1, 0)	(0, 0)	(0, 0)	(0,+1)	(0,-2)
2	(-1,+1)	(+1,+1)	(-2,+1)	(+1, 0)	(-2, 0)
L	(0,+1)	(0,+1)	(0,+1)	(0,-1)	(0,+2)

O Tetromino Offset Data



	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5
0	(0, 0)	No further offset data required			
R	(0,-1)				
2	(-1,-1)				
L	(-1, 0)				