FIT2102

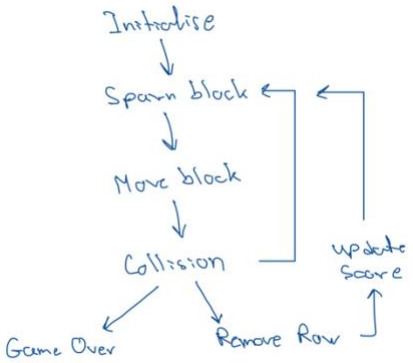
ASSIGNMENT 1

Shlok Arjun Marathe

Monash University Malaysia

Code Summary

- Developed in TypeScript using the FRP paradigm
- Used RxJS and Observables for creating an asynchronous and event-based program
- Notable parts:
    - The 7 different possible block types
    - Increasing difficulty by placing obstacles on grid at each level
    - Ability to restart by pressing space
    - Rotation system for the blocks
        - I did not use any pre-defined system; I just chose the seconds brick in the block and rotated the block about that brick
- Controls:
    - A/D to move the block left/right respectively
    - S to move the block down
    - W to rotate the block
    - Space to restart the game (Only when game is over)
- Flow of game:



Design Decisions:

- RNG Class for pseudorandom order of blocks
    - To make the code pure, i.e. not have side effects, I used a class for RNG as shown in the tutorial 4 exercises
    - Changed it to take in a seed and made a function nextInt to get the next integer
    - Only 1 seed is used
        - Ensures that the output is the same every time, which makes it pure
- Array for collision management linked to state
    - I added all the blocks that had been generated to an array stored in the state
    - This way, collisions can be checked for by comparing the x and y values of the current block with all the ones in the array
- Singular game state which is immutable
    - I used a single game state for the entire game
        - Stores gameEnd, currentBlock, nextBlock, score, level, highScore, existingBlocks
    - I chose this design so that there are no side effects that cause complications and the program is easy to test/predict
    - The only way to update the state is to replace some of the attributes of the state after performing actions

- Brick rotation
    - Rotates 90 degrees anticlockwise
    - I did not use any pre-defined system; I just chose the seconds brick in the block and rotated the block about that brick
    - If the brick is against a wall or next to another block, it won't be allowed to rotate
- Implementation of obstacles
    - At first, I was going to increase tick speed every level
        - Wasn't able to work out the logic for that
    - Using pre-defined obstacles
        - Every level, they appear one by one and the obstacles loop once there aren't any left
    - If there is already a block present at the location where the obstacle is to be placed, the obstacle won't be placed

Functional Reactive Programming Style & Observables:

- Observable Streams for Input
    - Instead of using event listeners, I've used observable streams to take all input from user to make use of FRP
        - Movement, rotation, restart
- Real time responsiveness
    - Subscribing to the observable stream allowed the blocks to render as soon as there is a change to the state
    - Ideal for games
- No variables
    - Ensuring that FRP style is followed and there is no imperative data
- Use of operators and observables
    - For a reactive codebase
    - Merge, filter, fromEvent, interval, map, scan

State Management:

- Made the state immutable
    - When a change occurs, new state is formed which replaces the old state
    - Old state remains unchanged, hence maintains purity
    - Able to predict output and the same input gives the same output
- Functions are pure
    - The functions I made do not modify any data in any way
    - The same input WILL always result in the same output
    - The move, rotate and restart functions change a state by returning a new state

Observable beyond simple input:

- Block moves down by itself every tick
    - The tick observable 'pulses' every game tick, which is pre-defined
    - This is caught in the main function and moves the blocks down
    - Observables have to be used in this case so that the state constantly updates and renders

Acknowledgements and disclaimers:

- The idea of an RNG class was taken from the FIT2102 tutorial 4 main.ts file
    - I adapted the code to work for my game
- All the code in all files was formatted using ‘Prettier - Code formatter’ on VS code