FIT 2102

Assignment 1 Report (Tetris in FRP Style)

Name : Chuah Sze Ler

Student ID : 33366586

Overview

- Functions have been broken to many small functions which helps to improve readability and reusability and easier to be analysed. (FRP Programming)
- Ternary Operator is widely used instead of if-else statement
- HOF is used in drawing tetrominoes to canvas
- Special observables are used when creating pause event

State and Managing Function

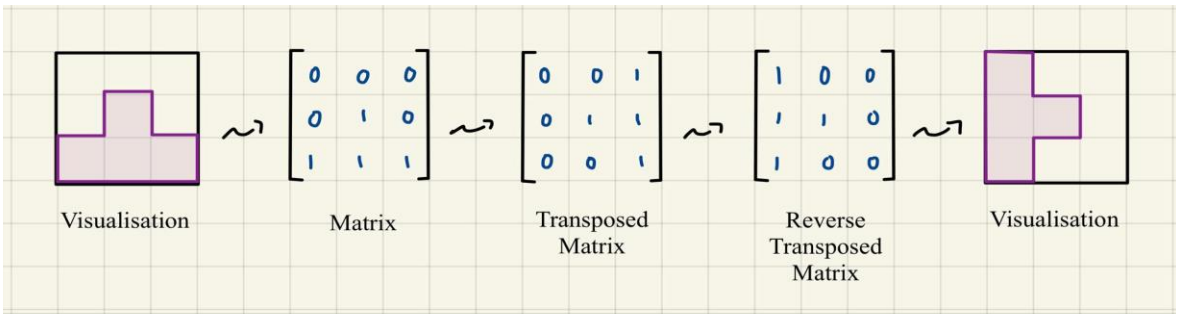
- Important States
    - **store** : 1D array used to store placed tetrominoes yet to be removed
    - **stayTime** : Hold the tetromino for Constants.Tick_Rate_MS*Max_Stay_Time prior dropping automatically
    - **yReach**, **xLeftReach**, **xRightReach** : Check bound of grid.
- To maintain purity of FRP, game starts with initialState and new state will be created tick by tick to show current movement of tetromino.
- Once yReach is true, store will be applied updateBlock function to update the position of current tetromino to the canvas

Generate Block (Lazy Sequence)

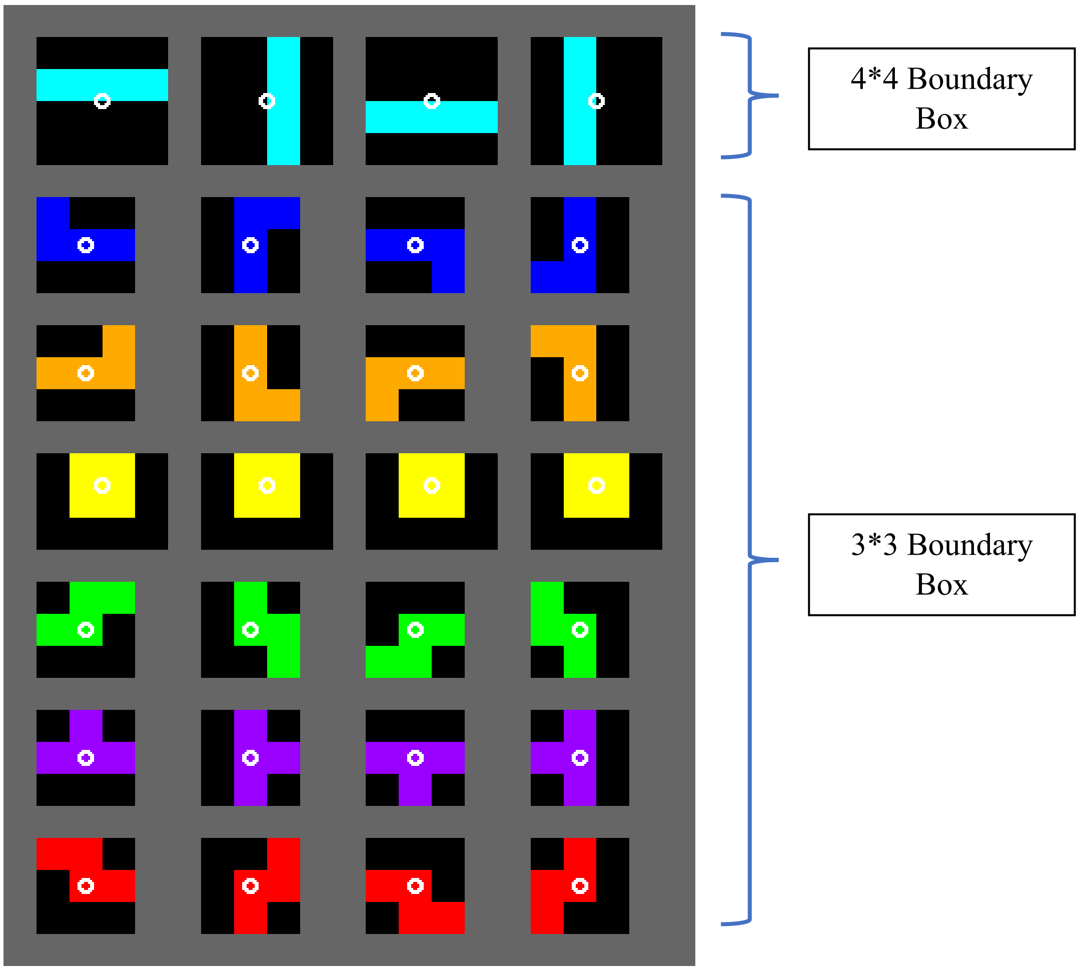
- Type **Block** is used to create the initial state of 7 different tetrominoes.
- Constant **tetrominoes** is then used to store a list of tetrominoes.
- Lazy sequence **seedGenerator()** is used to generate random sequence of tetrominoes.

Rotation System

- SRS rotation system is used to rotate each tetrominoes (**Clockwise** rotation take into consideration)
- Rotation is implemented by transposing and reversing of matrix (Example as below)


Visualisation
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Matrix
$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$
Transposed Matrix
$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
Reverse Transposed Matrix
Visualisation

- Each tetromino rotation follows below



Difficulty System & Scoring System

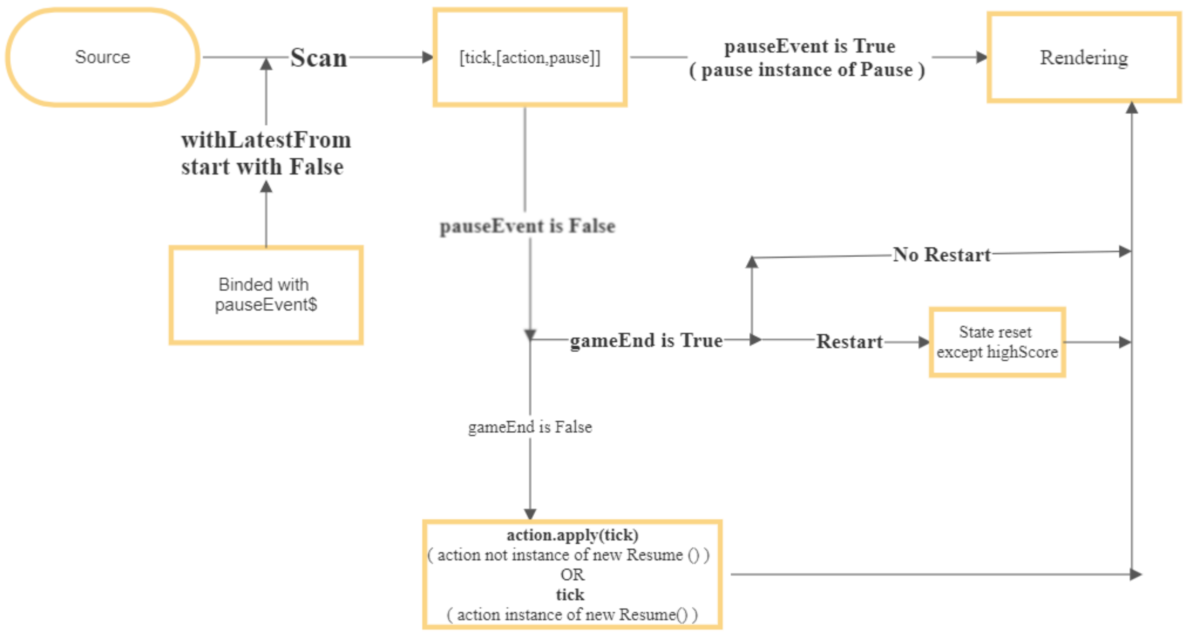
- Tetromino start dropping with Constant.Tick_Rate_Ms*MaxStayTime which is **(50*20)ms**
- When lineClear accumulated the total of **>= 5** ,
  - dropping speed updated to **(50*(MaxStayTime-1))ms,** n as current MaxStayTime.
  - lineClear return lineClear%5
  - level increases by 1
  - Score increases by 1000 for each line cleared.
  - HighScore will only be updated if current round has a score larger than HighScore.

Render (HOF)

- Impure functions are found here which modify DOM element
- Canvas is cleared using **clear** function which sets innerHTML to **svg.lastchild** (Clear all unwanted DOM, ensuring next tick will have accurate presentation for everything)
- **gridDraw** function is used to draw the grid back to canvas
- Level, Score, HighScore is updated
- **drawTetromino** (HOF) , returns a function which is used to draw tetromino on Canvas. It is used to draw **current tetromino**, **tetromino found at store** and **next tetromino to be generated**.

Observable

- Constant **keyboard** combines all keyboard event
- Observable stream starts with merging keyboard stream & tick stream
- **withLatestFrom** is used to combine the latest values emitted by merge and pauseEvent$ observables.
- Scan operator is then used to apply action emitted by merge to current state of game
- If latest value emitted by **pauseEvent$** is Pause, game state will not be updated.
- If latest value emitted by **pauseEvent$** is Resume, game state will not be updated but resume the game
- Otherwise, game state will be updated with the action applied to it.
- Subscribe function check if the game has ended
- If gameEnd is true, gameover element appended and shown.
- Diagram below shows how it works



Game Restart

- Whenever state gameOver returns true, the whole game halts and gameOver will be shown
- To restart the game, key Restart is pressed to restart the whole game with states updated to initial state except for state HighScore which will take in the current high score from last game.

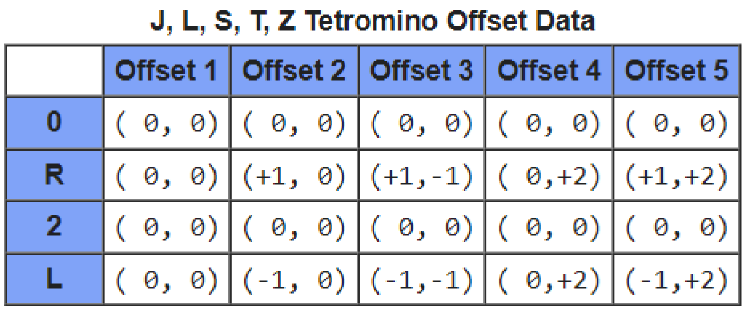
Game Control

- Key A – Left
- Key D – Right
- Key W – Rotate
- Key S – Down
- Key P – Pause
- Key R - Resume
- Key Space – HardDrop
- Key Enter – Restart

Advanced Features

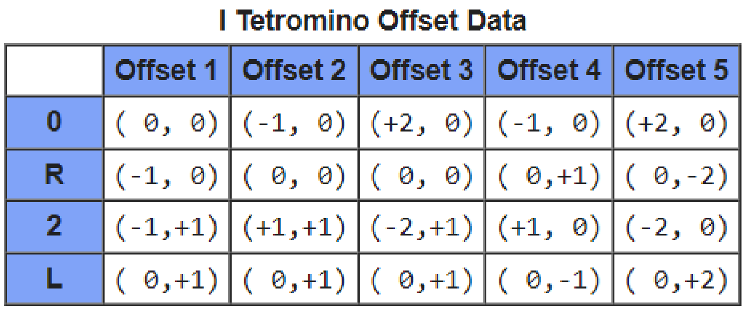
<u>WallKick</u>

- SRS Rotation System helps to implement wallkick with ease.
- Wallkick is a mechanism which allows tetromino to rotate even though its pending rotated place is occupied (Either reaches bound / Collide with other tetrominoes) by kicking it out to an empty position
- When testing wallkick, table below is used where each rotation state has 5 tests.



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)

- If all 5 tests fail, the tetromino can't be rotated
- When **Rotate** is called, constant rotated block will be used store the rotated tetromino
- Constant **wallKickType** (2D Array of wall kick data of current rotation) will be used to determine which table above should be used. If is I tetromino, table below will be used, else table above will be used.
- Constant **tryWallKick** is used to check if the rotation after wallkick is valid
- Constant **validRotate** will then use wallKickType and map each tested position to the current position of the tetromino after rotating. find function will then be used to find the first valid rotation after wall kick.
- Return the updated state if wall kick is success.
- If wall kick fails, return the state without updating any states.

HardDrop

- With hard drop, user can place current tetromino to the lowest possible position swiftly.
- Constant **dropVal** is used to increase value by 1
- Constant **checkCollisionAndBoundary** uses the combination of **boundaryCheck**, **xCollision**, **yCollision** to check for each increased y value will they return true or false
- Constant **finalDropVal** will then use an Array (ranged from 0 to Constant.**Grid_Height** – current y value) and map to dropVal in order to increase values in the array. Then, it maps the result to checkCollisionAndBoundary and finally uses **indexOf** Observable to return the first index which returns true, indicating no obstacles and no tetromino will be above the bound.
- Return the updated state

Pause Event (Special Observable Method)

- Pause Event is implemented with **withLatestFrom** and **StartWith** operator from rxJs.
- **withLatestFrom** is used to combine the latest operation emitted by merge and pauseEvent$ observables.
- The main stream will then bonded with the combined operation.
- **pauseEvent$** merges pause$ and resume$ keyboard event (Key P and Key R), theses 2 keyboard event are bonded with a Boolean indicating the pause or resume of the game
- **startWith** is used to initialise a false operation into it (new **Resume()** here) .
- **scan** operator is then used to apply actions emitted by merge.
- If pauseEvent$ is triggered by **KeyP**, state pause will be modified to true and the game will be paused.
- If pauseEvent$ is triggered by **KeyR**, state pause will be modified to false and the game will be resumed.