FIT 2102 Assignment 1 Report

Name: Mah Ying Qi

Student ID: 32796765

Summary of Tetris Game Implementation

- Use of observables
    - The use of observables in my assignment is similar to the asteroid example
    - Moving left, right and down are allowed to repeat
    - Rotating is not allowed to repeat to follow the original gameplay.
    - The key entries and tick are merge into one observable and handled together
    - Tick interval is set to 0 so that the limit of interval of tick is minimum
- State management
    - States are managed such that functions take old state as parameter, then create a new state that is based on the parameter and process some value.
    - This will maintain the purity as it does not mutate the parameter state
- Tetromino is named as shape, and single cube is named as block in my code and report
- Used a 10*22 matrix to record positions occupied
    - Easier to validate whether the shape can be moved to that position
    - Faster than looping through all the blocks in the canvas every time
    - 22 for height so that the shape can be spawned on top of the canvas
- How the shape is handled
    - Current shape is implemented such that they are 4 pieces of blocks held together
    - This grid implementation is easier to handle than a single block that is 4 times the area of a block.
- Algorithm of clearing full rows.
    - First find the full rows, then remove all the blocks on rows that are full, then update the matrix.
    - After clearing, drop the rows on top of the cleared rows according to how many rows are cleared below them.
- Shape is spawned on top of the canvas
    - So that game will end when a shape landed out of the canvas.
    - At row of height 21 or 22
- Handling shape landed
    - Check if shape has landed using the 10*22 matrix by checking if there's a stationary block under current shape.
    - When shape has landed, put it in stationary block array, make the preview as current shape, and create a new shape for preview.
- Shape moving
    - Moving left, aright and down are performed in the same function to reduce duplication of code
- Handling game end
    - Checks if there is a landed block out of the canvas using the 10*22 matrix
    - Remove all the blocks
- Restarting game
    - Create a new game state but preserve the high score and blocks count.
    - Preserve blocks count so that the new object id does not overlap with the old blocks
    - Preserve the high score to keep track of it in the session
- Object ID for each block.
    - Create an object id based on the name of the shape, and the number of blocks to prevent overlapping of id for DOM element
- Rotating system.
    - SRS- rotating system is implemented
    - All the orientation of blocks of shapes are prelisted in an array and use a rotate state number to determine which is the next shape to use.
- Shape preview to current shape
    - Store preview shape

    - When a shape has landed, create a new shape based on the preview shape, and set the position to the spawning position in the gameplay canvas.
- Random shape selecting
    - All the shape creating functions are stored in an array, and a hash function is used to hash the object id to a random number below the length of array to determine which function to use.
- Increases difficulty after row clearing
    - Increase dropping speed of shape
    - The interval number of the tick is passed to the tick function
    - The speed is controlled by saving a speed in the game state, so when the interval number has reached a multiple of the speed, for example the speed is 200, and the interval number is 600, which results in 600%200 = 0, the tick function will drop the shape by 1 row.
    - The dropping speed is increased by decreasing the speed attribute in game state. For each level up, the speed number will be decreased by 20.
- Shape rendering
    - Used .map() to change attributes, append or remove the elements to canvas or preview box
- Additional Requirement: Landing delay
    - Shape will delay for 2 ticks before becoming stationary.
    - Clicking S when landed will shorten the delay by 1 tick.
    - This is achieved by adding a landing delay value in the state, and decrease the landing delay by 1 when tick happens, also when S is clicked
- Additional Requirement: Wall kicking
    - When the position after rotating is not valid, the shape is moved right first, if the position after moving right is valid, then place the rotated shape at right side
    - If not valid, try left, if still not valid, rotating fail.
    - Achieved by creating the rotated shape first, then check if it is valid using the 10*22 matrix
- Additional Requirement: Shape storing
    - 2 attributes (store, changedStore) are added to the state type, 1 attribute (swapped) is added to the shape type to achieve this
    - When key C is clicked, the current shape will be placed in the store canvas, and current shape will be replaced with the stored shape.
    - If there's no stored shape, the preview shape will be used to replace the current shape.
    - Swapping can only be performed once for each time landing.
    - Box on top is preview, box below is store