Assignment 1 FIT2102 - Tetris
By Caleb Smith

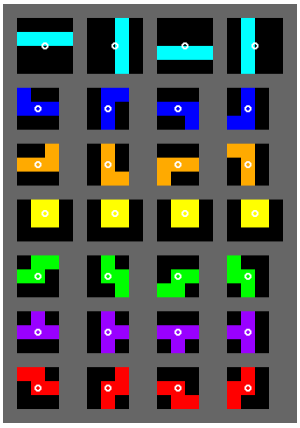
Key Decisions:

Block Creation

The block and units were custom types to help keep the information about them ordered and correctly used. Units are a single square and a block has an array of units which gives it its shape. The blocks also need a center that they can then rotate around. To create all the different types of blocks easily I used a curried function that takes in block color, unit coordinates and the center of the block and which returns a function that given an id will return a block of that type with that id. This allows me to use an array with each element being the partially applied block creator for that type and using an index in that array will return the function to create that block. This allows for easy extension through only needing one line to add in additional blocks by adding the partially applied block creator to that array.

Block Movement

To organize all the movements of a block I added a custom moveFunction type that defines a function that inputs and outputs a block after moving it some way. I used a higher order function createMoveFunction that takes in two functions, the mapping of each unit in the block and the center of the block. This was a powerful tool that allowed me to create a function with any kind of movement on a block. I then created the move functions moveX, moveY, rotateLeft and rotateRight. This allows for all the needed movement within the game and moveX and Y take in an amount input to allow them to move the block a variable amount in that direction.



Rotation system used utilizing the center point

Randomisation

Randomized blocks through a custom observable that take inspiration from tutorial 4 with the RNG stream. Each tick takes in a seed, hashes in and returns the scaled value from 1-7. 1-7 correlates to the create block function in the block creators array, and will return the function to create that specific block. Whilst the RNG creation through the observable is pure, the creation of the starting seed uses a randomSeed const which is set by math.random(). This is used so that every game has a different starting seed and the sequence when starting a new game cannot be memorized. The randomseed is also used in determining the starting and previewed block in the initial state.

Game Progression

The game uses an action interface as seen in asteroids example. This allows each user input to map to an Action class implementation which has the apply function so the action can be applied. This means that each observable which checks for an input is merged together, along

with the RNGtick observable. The game uses a reduce state function to apply the action, and will return the next state within the game. This allows the state of the game to keep changing through each action and is stored within the observable.

Game Layout

The game state type contains:

- level, score and highscore to keep track of game stats
- Next block, to allow for the preview to be shown
- Current block, to store the current block after each movement
- Gamegrid, to store an array of all the units that are placed in the game grid
- Game end boolean to determine if the game has ended
- Tickspeed which determines how many milliseconds between frames
- Past state, to help remove the old resources from view each frame

The game grid decomposes the block into its units, as once it is placed on the game grid the block does not move together anymore.

Advanced features:

- Game speed curve based on level:
  The game uses the level to calculate milliseconds between frames. This is approximated using an exponential decay curve mapped to be close to the original tetris. The ticks are then filtered out to only run ticks that are multiples of the current tick speed. This leads to every 1 in tickspeed frames being run, allowing use to dynamically change the game speed to be faster as each level passes
- Left and right wall kicks
  Wall kicks are introduced on rotations, where a rotation fails on a collision, it will then attempt to complete that rotation by moving the block left or right. If successful the block will be ‘wall kicked’ so it can rotate
- Soft drop
  The down key soft drops the block so that it is dropped all the way down but is not placed. This can be convenient in gameplay as you can speed up the falling of the block but still manipulate it right before it is placed
- Hidden blocks shown their future position in grey
  In the view portion of the code, any block that is rendered above the grid will now display a grey preview of where the block will be when it does drop into the grid as blocks start above the grid. This allows for accurate game over detection as blocks are attempted to place down on a full row and will end the game. It also gives the player information about where the block is, what rotation and block shape when they can't fully see the block on the grid.
- Unplaced blocks have a lower opacity to indicate they aren't place
  Through the use of opacity, the current block's opacity is lowered to indicate that it has not been placed yet. This was needed in order to help soft drops appear functional as upon soft dropping, the block will still have the lowered opacity indicating that it can still be moved.