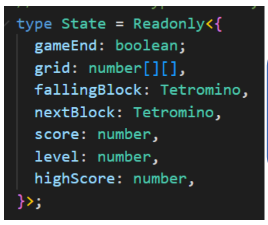
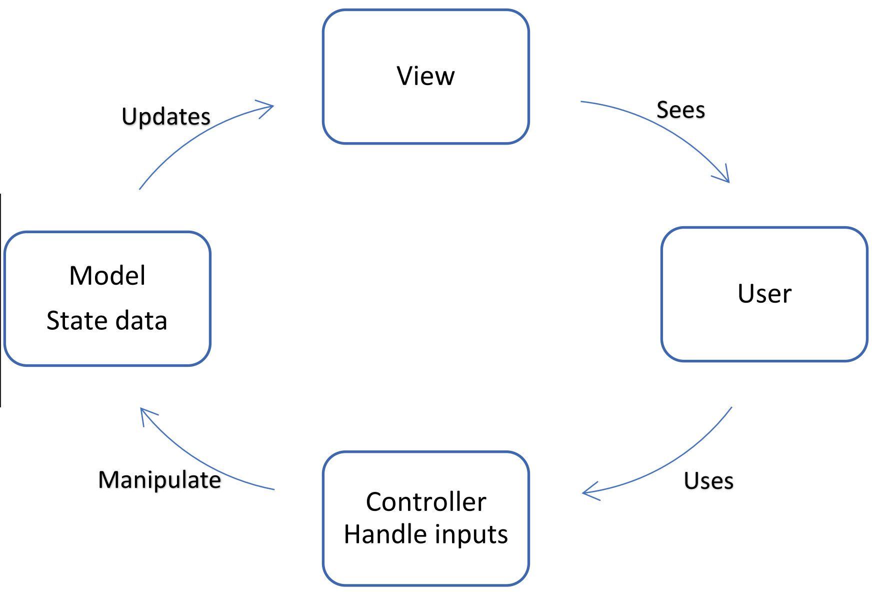
Code Overview

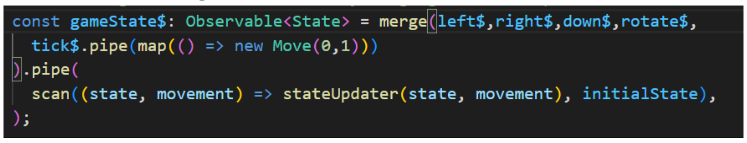
I have followed the Model-View-Controller architecture to code in functional reactive programming style. I separated state management, key inputs, visual updates, and combining them all into one merged observable stream.



```
type State = Readonly<{
  gameEnd: boolean;
  grid: number[][],
  fallingBlock: Tetromino,
  nextBlock: Tetromino,
  score: number,
  level: number,
  highScore: number,
}>;
```

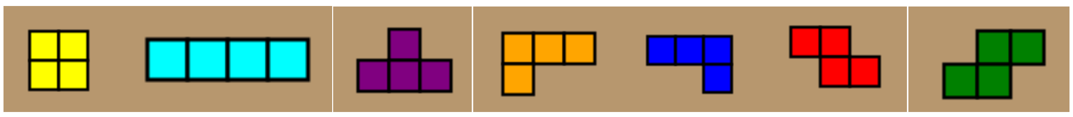


This is all tied together with an observable stream.



```
const gameState$: Observable<State> = merge(left$,right$,down$,rotate$,
  tick$.pipe(map(() => new Move(0,1)))
).pipe(
  scan((state, movement) => stateUpdater(state, movement), initialState),
);
```

- Constants: Constants are used to define game parameters, such as grid dimensions, tick rate, and block dimensions, making it easy to adjust these values if needed. This will also mitigate the side effects.
- Tetrominoes: An array of Tetromino objects is defined to represent the different types of tetrominoes used in the game. These objects contain shape, color, and position information.



FRP Style and Observable Usage

1. **Observable Streams**: Various Observables are created for user input events, such as key presses (left, right, down, rotate), and timer ticks. I have also created KeyObservable function to reduce the repetition of creating the same type of Observables from keypress events. These streams are merged to create a single stream of events for the main game stream.
2. **State Updates**: The scan operator is used to accumulate state changes over time. It takes the current game state and movement (which is user keyboard events), applies a state update function, and produces a new game state. This ensures that the game state is updated reactively in response to user input and timer ticks (500ms).

State Management

In this Tetris game code implementation, state management is achieved by:

- Creating an immutable initial game state with an empty grid and initial tetrominoes.
- Ensuring that all state updates are done through pure functions, such as stateUpdater, which calculates new states based on user input and enforces game rules.
- Avoiding direct manipulation of the game state (using the spread operator), which could lead to side effects.

Movement control

Movement control was achieved by (this was derived from asteroid game and week 3 tutorial):

- Timer ticks from the tick$ Observable handles the downward movement of the current falling tetromino.
- The merging of different input Observables allows for dynamic responses to user actions, such as block rotation or movement.

Tetromino rotation

- I have chosen to implement my own rotation system due to the complexity of provided rotation systems.
- My rotation system works by transposing and reversing the rows and columns while using a new constant to avoid mutation.
- I have tried to implement Arika/TGM rotation system, but with the time constraints, I decided to opt out.

Collision handling

Constants are created to avoid mutating variables.

- Before applying movement action to the current falling tetromino, collision detection functions are used to check if the tetromino can move in desired direction.
- If a collision is detected (walls or blocks), the movement is restricted.

Locking tetrominos

- Functions are created to check if a tetromino should be locked in place.
- Tetrominoes becomes locked in place when it has reached the bottom or collides with other tetrominoes. Locking the tetromino ensures that it will float on top of each other.

Game over

- When a tetromino block reaches the top of the canvas, the game ends. However, it only ends if user presses any key (except for 'KeyS').
- I have tried to implement several ways to get game over to work but I have failed all the time.
- What I've tried:
  - If the top row has any occupied grids, the game ends.
    - Did not work because all blocks must go through the top row…
  - Full column leading to game end.

- Will not work because the column might not be full for a tetromino to touch the top of canvas.