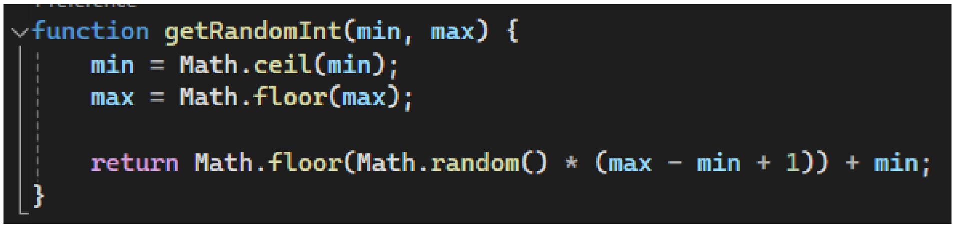
Tetris Documentation: Spring 25 –Izzy

1. Frontend: Game Logic

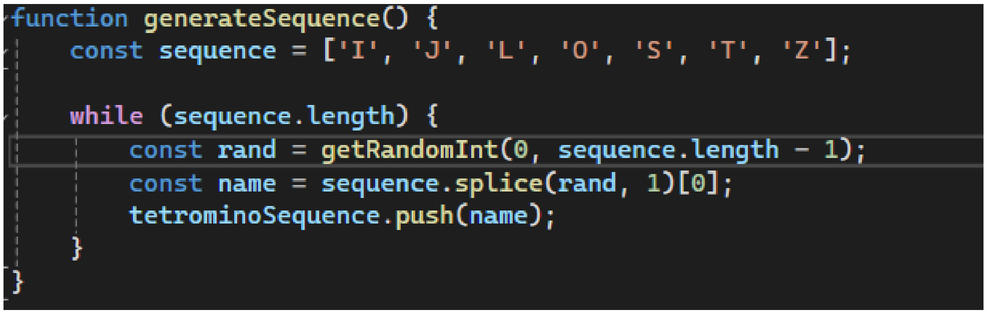
The JavaScript game uses the jQuery library. The logic handles the gameplay mechanics and interaction with the grid. Key methods include:

1. getRandomInt():
   - method takes in 2 parameters; it then sets a floor and ceiling. Once done it find a random within the values given values.



```
function getRandomInt(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);

    return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

2. generateSequence():
   - This is just their take on the random() for the Tetris pieces.
   - This method creates a sequence of letters. Dose a while loop were getRandomInt(): takes in the min value 0 and max of seq.len -1. It then chooses a random letter from sequence and removes it and adds it to another array [0]

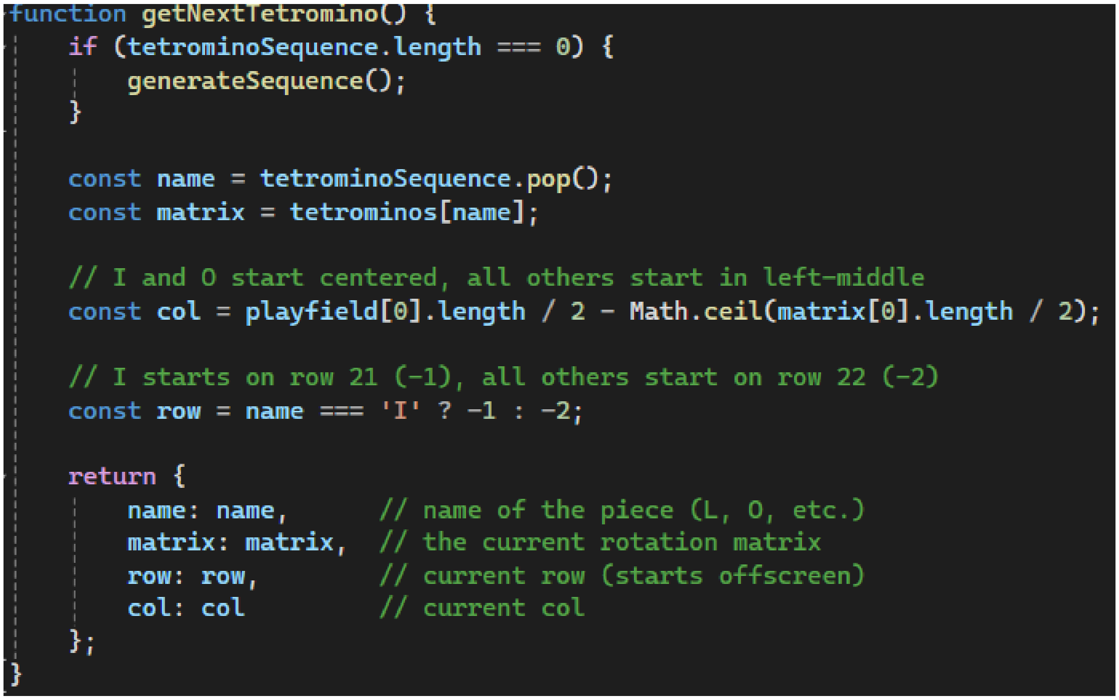


```
function generateSequence() {
    const sequence = ['I', 'J', 'L', 'O', 'S', 'T', 'Z'];

    while (sequence.length) {
        const rand = getRandomInt(0, sequence.length - 1);
        const name = sequence.splice(rand, 1)[0];
        tetrominoSequence.push(name);
    }
}
```

3. getNextTetromino():

- This method start by checking if terominoSequence is empty. If it is, call the last method generateSequence for values.

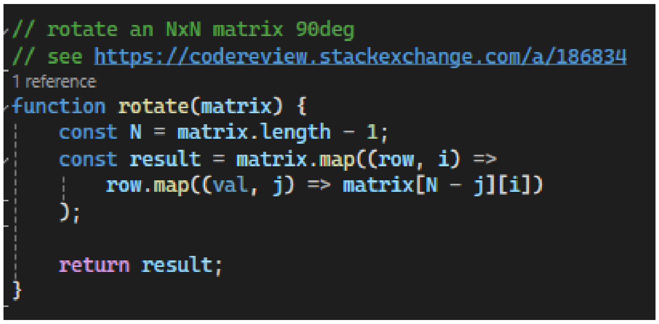
- Else, it will pop the last item in the array terominoSequence and return the value that will be used in teromino to determine the sequence. Basically, it returns one of the letters e.g. “L” that is associated with its matrix pattern. The rest in already commented.



```
function getNextTetromino() {
    if (tetrominoSequence.length === 0) {
        generateSequence();
    }

    const name = tetrominoSequence.pop();
    const matrix = tetrominos[name];

    // I and O start centered, all others start in left-middle
    const col = playfield[0].length / 2 - Math.ceil(matrix[0].length / 2);

    // I starts on row 21 (-1), all others start on row 22 (-2)
    const row = name === 'I' ? -1 : -2;

    return {
        name: name,      // name of the piece (L, O, etc.)
        matrix: matrix,  // the current rotation matrix
        row: row,        // current row (starts offscreen)
        col: col         // current col
    };
}
```

4. rotate():

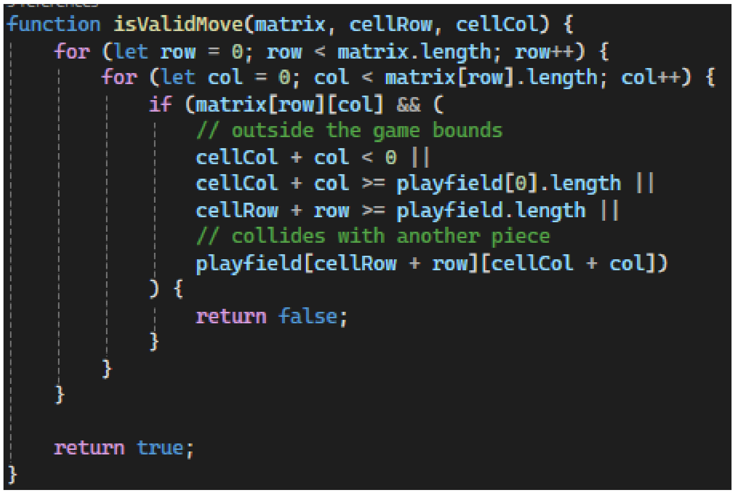
- Its in the Name



```
// rotate an NxN matrix 90deg
// see https://codereview.stackexchange.com/a/186834
1 reference
function rotate(matrix) {
    const N = matrix.length - 1;
    const result = matrix.map((row, i) =>
        row.map((val, j) => matrix[N - j][i])
    );

    return result;
}
```

5. isValidMove():

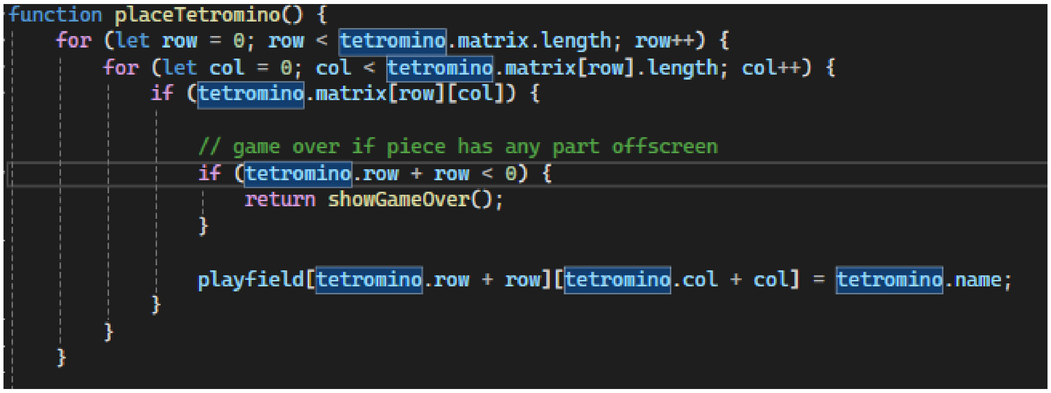
- This is game logic. The function determines if a tetromino can be placed at a given position without going out of bounds or colliding with another piece.



```
function isValidMove(matrix, cellRow, cellCol) {
    for (let row = 0; row < matrix.length; row++) {
        for (let col = 0; col < matrix[row].length; col++) {
            if (matrix[row][col] && (
                // outside the game bounds
                cellCol + col < 0 ||
                cellCol + col >= playfield[0].length ||
                cellRow + row >= playfield.length ||
                // collides with another piece
                playfield[cellRow + row][cellCol + col])
            ) {
                return false;
            }
        }
    }

    return true;
}
```

6. placeTeromion()

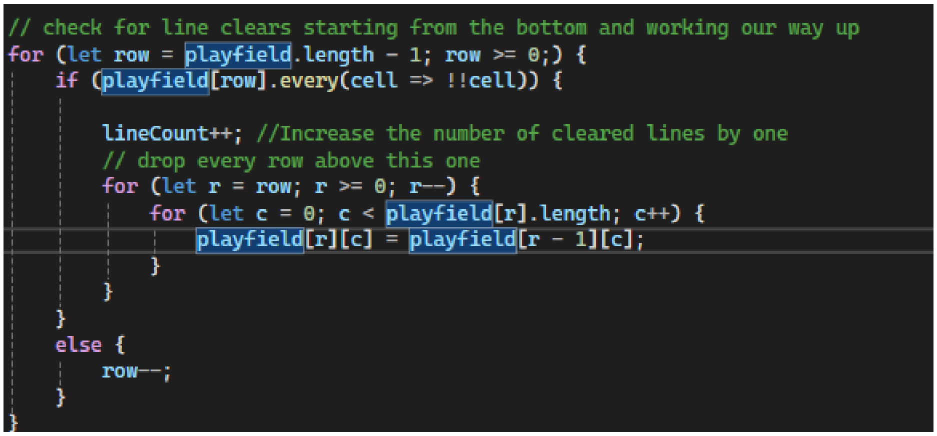
- Places the tetromino onto the playfield. If any part of the piece is above the screen when placed, the game ends. Updates the playfield array to reflect the tetromino's position.



```
function placeTetromino() {
    for (let row = 0; row < tetromino.matrix.length; row++) {
        for (let col = 0; col < tetromino.matrix[row].length; col++) {
            if (tetromino.matrix[row][col]) {

                // game over if piece has any part offscreen
                if (tetromino.row + row < 0) {
                    return showGameOver();
                }

                playfield[tetromino.row + row][tetromino.col + col] = tetromino.name;
            }
        }
    }
```

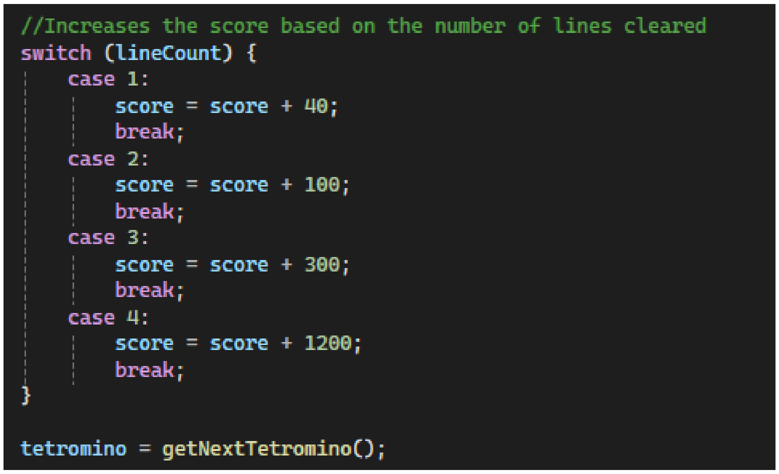
7. Check if row is full

- Starts from bottom and goes up checking if the row is full across. If it is, remove the row and drop the row above by one. Else, move to next row.



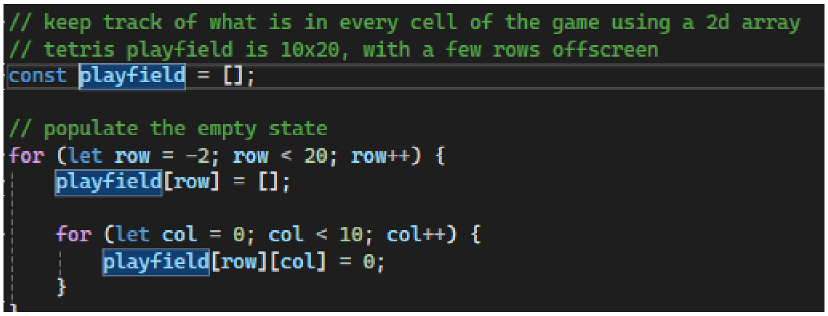
```
// check for line clears starting from the bottom and working our way up
for (let row = playfield.length - 1; row >= 0;) {
    if (playfield[row].every(cell => !!cell)) {

        lineCount++; //Increase the number of cleared lines by one
        // drop every row above this one
        for (let r = row; r >= 0; r--) {
            for (let c = 0; c < playfield[r].length; c++) {
                playfield[r][c] = playfield[r - 1][c];
            }
        }
    }
    else {
        row--;
    }
}
```

8. Get new piece and update score
    - I mean, come on. You understand.



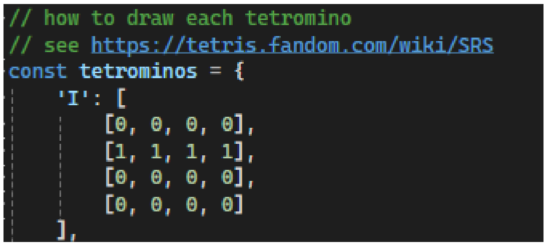
```
//Increases the score based on the number of lines cleared
switch (lineCount) {
    case 1:
        score = score + 40;
        break;
    case 2:
        score = score + 100;
        break;
    case 3:
        score = score + 300;
        break;
    case 4:
        score = score + 1200;
        break;
}

tetromino = getNextTetromino();
```

9. Making the Tetris board
    - Read the green comments. They got it.

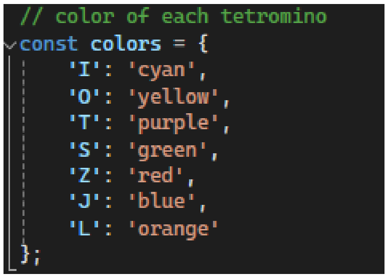


```
// keep track of what is in every cell of the game using a 2d array
// tetris playfield is 10x20, with a few rows offscreen
const playfield = [];

// populate the empty state
for (let row = -2; row < 20; row++) {
    playfield[row] = [];

    for (let col = 0; col < 10; col++) {
        playfield[row][col] = 0;
    }
}
```

10. Tetrominos
    - They made the shapes of all the Tetris pieces using matrices. This is one of them. You can look at the rest. I will put colors of them with this as well.



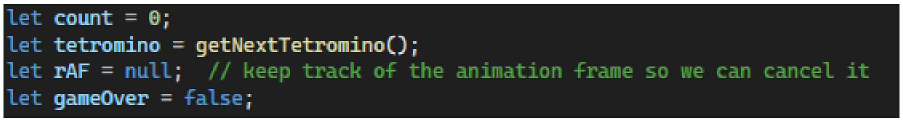
```
// how to draw each tetromino
// see https://tetris.fandom.com/wiki/SRS
const tetrominos = {
    'I': [
        [0, 0, 0, 0],
        [1, 1, 1, 1],
        [0, 0, 0, 0],
        [0, 0, 0, 0]
    ],
```



```
// color of each tetromino
const colors = {
    'I': 'cyan',
    'O': 'yellow',
    'T': 'purple',
    'S': 'green',
    'Z': 'red',
    'J': 'blue',
    'L': 'orange'
};
```

11. Setting initial values

    - Prior to starting the initial values are set



```
let count = 0;
let tetromino = getNextTetromino();
let rAF = null;  // keep track of the animation frame so we can cancel it
let gameOver = false;
```

12. Remaining

    - There are 2 methods left Loop and add even Listener.

        - Loop()

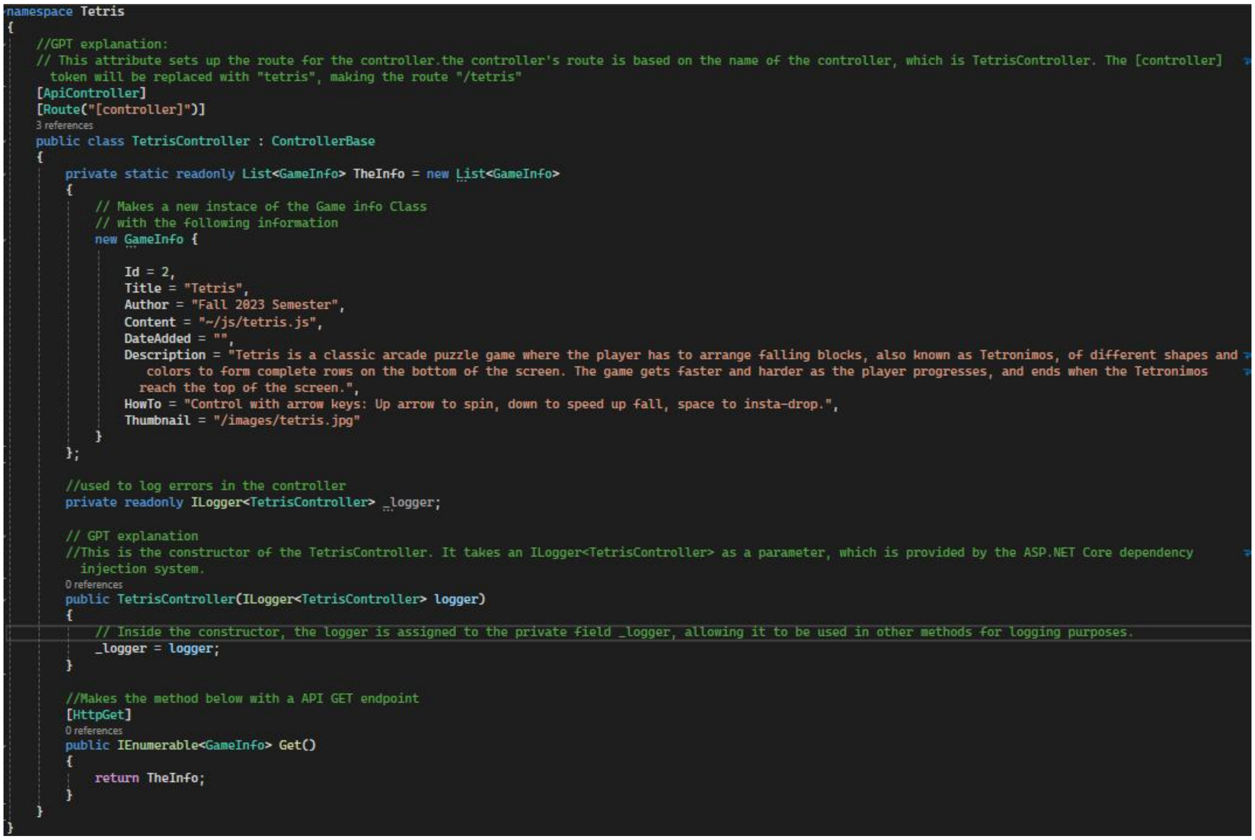
            i. This is the main game logic. They did a good job at commenting on this part so read the comments. Yull be fine.

        - AddEvenListener()

            i. This portion is how the keyboard strokes accomplish the movements for the game. They also did comment this well, so give it a read.

2. Tetris Controller Class

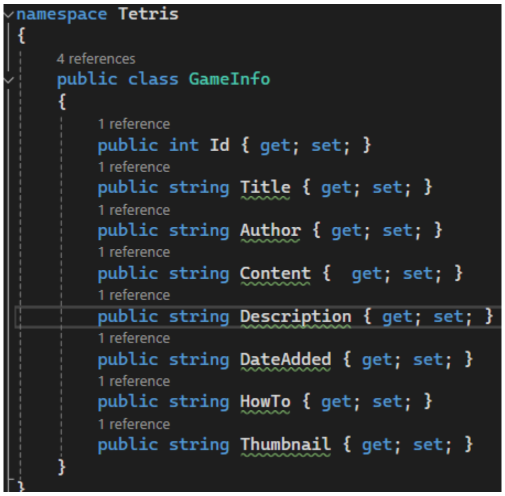
- This Game Info came up on the swagger website when I ran it, but not the actual game. I commented the code with the information.



```
namespace Tetris
{
    //GPT explanation:
    // This attribute sets up the route for the controller.the controller's route is based on the name of the controller, which is TetrisController. The [controller]
      token will be replaced with "tetris", making the route "/tetris"
    [ApiController]
    [Route("[controller]")]
    3 references
    public class TetrisController : ControllerBase
    {
        private static readonly List<GameInfo> TheInfo = new List<GameInfo>
        {
            // Makes a new instace of the Game info Class
            // with the following information
            new GameInfo {

                Id = 2,
                Title = "Tetris",
                Author = "Fall 2023 Semester",
                Content = "~/js/tetris.js",
                DateAdded = "",
                Description = "Tetris is a classic arcade puzzle game where the player has to arrange falling blocks, also known as Tetronimos, of different shapes and
                   colors to form complete rows on the bottom of the screen. The game gets faster and harder as the player progresses, and ends when the Tetronimos
                  reach the top of the screen.",
                HowTo = "Control with arrow keys: Up arrow to spin, down to speed up fall, space to insta-drop.",
                Thumbnail = "/images/tetris.jpg"
            }
        };

        //used to log errors in the controller
        private readonly ILogger<TetrisController> _logger;

        // GPT explanation
        //This is the constructor of the TetrisController. It takes an ILogger<TetrisController> as a parameter, which is provided by the ASP.NET Core dependency
          injection system.
        0 references
        public TetrisController(ILogger<TetrisController> logger)
        {
            // Inside the constructor, the logger is assigned to the private field _logger, allowing it to be used in other methods for logging purposes.
            _logger = logger;
        }

        //Makes the method below with a API GET endpoint
        [HttpGet]
        0 references
        public IEnumerable<GameInfo> Get()
        {
            return TheInfo;
        }
    }
}
```

3. Game Info Class

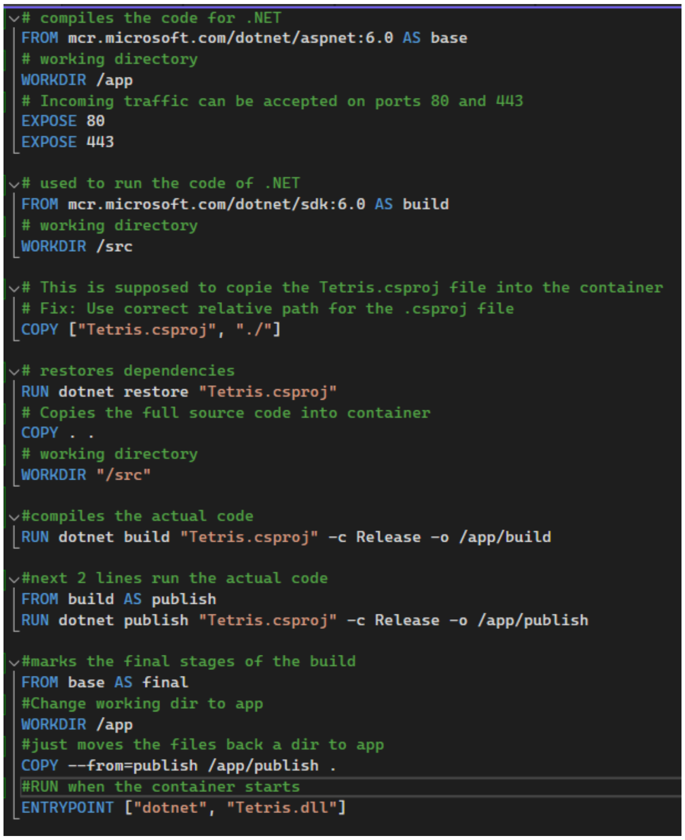
- Holds game metadata such as title, description, and instructions for playing. This class is used to provide a quick overview of the game. This is not used in the JavaScript only in the controller



```
namespace Tetris
{
    4 references
    public class GameInfo
    {
        1 reference
        public int Id { get; set; }
        1 reference
        public string Title { get; set; }
        1 reference
        public string Author { get; set; }
        1 reference
        public string Content {  get; set; }
        1 reference
        public string Description { get; set; }
        1 reference
        public string DateAdded { get; set; }
        1 reference
        public string HowTo { get; set; }
        1 reference
        public string Thumbnail { get; set; }
    }
}
```

4. Tetris Docker File

- I went ahead and just commented on all the for this section as I thought it would be easier this way.



```
# compiles the code for .NET
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
# working directory
WORKDIR /app
# Incoming traffic can be accepted on ports 80 and 443
EXPOSE 80
EXPOSE 443

# used to run the code of .NET
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
# working directory
WORKDIR /src

# This is supposed to copie the Tetris.csproj file into the container
# Fix: Use correct relative path for the .csproj file
COPY ["Tetris.csproj", "./"]

# restores dependencies
RUN dotnet restore "Tetris.csproj"
# Copies the full source code into container
COPY . .
# working directory
WORKDIR "/src"

#compiles the actual code
RUN dotnet build "Tetris.csproj" -c Release -o /app/build

#next 2 lines run the actual code
FROM build AS publish
RUN dotnet publish "Tetris.csproj" -c Release -o /app/publish

#marks the final stages of the build
FROM base AS final
#Change working dir to app
WORKDIR /app
#just moves the files back a dir to app
COPY --from=publish /app/publish .
#RUN when the container starts
ENTRYPOINT ["dotnet", "Tetris.dll"]
```