2:GameLogic.cs

2.1:General Overview:

GameLogic.cs contains the GameLogic class. The GameLogic class contains most aspects of the game that do not fit somewhere else better.

The majority of functions in the program interact with GameLogic in some form

2.2:Variables:

There are too many variables to explain each one individually, but the major ones will be noted

ActiveTetomino curtet: this is an ActiveTetromino variable representing the current active tetromino. This is used very very frequently.
static byte[][] board: this is the variable that stores the data of the game board. It uses the TetrominoID numbering system (3.2) to store the values, and is the primary reason why 0 is empty instead of a tetromino in the system. It is a jagged byte array that is 20x10. The first number is the y position, and the second number is the x position (they were initially the other way around but had to be swapped in order to make line clearing work smoothly). The top of the board is index 0 while the last row is 19. The x positions are stored left to right.
Randomizer randomizer: this is the implementation of the randomizer class in the game. All random functions are called through this.
DispatchTImer gravtimer: the dispatch timer for the gravity function. It is turned on and off based on the circumstances.
DispatchTimer lockdowntimer: the dispatchtimer for lockdown. When the timer ends the tetromino locks down, but moving or rotating the tetromino resets the timer
Static byte highestfillrow: this variable keeps track of the highest row on the board that has a solid mino in it. It used used to save processing time on various functions.
Static ints level, score, and totalcleared: these are variables that keep track of the current leve, the score, and the total number of lines cleared respectively
Static byte holdtype: stores the TetrominoID (3.2) of the tetromino in hold.
Static int[] scorevals: this is an array of points received for various actions. The first 5 are for clearing 0-4 lines, then the next 2 are for a mini tspin clearing 0 or 1 lines, and the last 4 are for tspins clearing 0-3 lines

2.3:Functions;

Constructor: does nothing other than create the object.

startGame: resets most variables to their starting values and then starts the game. Called when starting a new game.

testLineClear: this function tests if the tetromino just locked down clears any lines. Clearcheck and passedclear variables are used to store the rows to test as well as their results. Sets numclear to the number of lines successfully cleared and returns that number

clearLine: line clear function. If anything in GameLogic is going to break, it will be this function. It is frankly a mess that gets the rows of the cleared lines into an array, sorts that array from largest to smallest, then loops through every row between the lowest row(highest number in array) and highestfillrow and copies the data from the row that it will end at. It increases the level and gravity if it has passed the required threshold. It then does some math to calculate the score to add and adds it.

Gravity: this is the function called by gravtimer. Self explanatory and easy to understand at a glance

lockdown: this is a very simple function that triggers after the lockdown timer is up.

Gameover: ends the game. Will crash if left or right haven't been used this game, but it was an extreme edge case that I didn't have time to fix

3: TetrominoData.cs

3.1: General Overview:

This class is almost entirely used to define variables that are constant. Use the tetdataArr to access all you need. See the comments in the file for descriptions of the variables. The 2nd constructor is for Standard SRS points.

3.2: TetrominoID Explanation:

TetrominoID is the way I describe bytes who's values range from 0-7 and are used to represent specific tetrominoes.

ID	Piece Shape	Color	Uses Standard SRS?
0	EMPTY SPACE	Black	N/A
1	I	Light Blue	NO
2	T	Purple	YES
3	O	Yellow	NO
4	J	Dark Blue	Yes
5	L	Orange	Yes
6	S	Green	Yes
7	Z	Red	YES

4: ActiveTetromino.cs

4.1: General Overview:

Easily the longest code file in the project. Represents the ActiveTetromino and various functions and variables that go with it.

4.2:Variables:

As with GameLogic.cs only major variables will be described here

Byte minotype: stores the TetrominoID of the current tetromino (3.2)

Sbyte positionx and positiony: the x and y positions on the board for the origin of the mino positions specified in TetrominoData

Sbyte ghostpos: y position of the ghost piece. Refreshed on various actions. Can be used to calculate if the tetromino is on the ground

Byte rotation: 0-3 storing the facing direction of the tetromino. Starting at north and going clockwise

Sbyte[] minoposx, minoposy: stores the positions of each mino in the active tetromino on the board. Calculated based on TetrominoData.

Bool istspin, ismini: is the tetromino currently in a tspin or tspin mini state respectively

4.3:Functions:

Constructor: basically the same thing as NextActive but for constructing the class

TryMoveX: tests if the tetromino can move in the specified direction (true for left false for right)

TrySoftDrop: this function is used for finding the position of the ghost piece. Was originally used for soft dropping too, but softdropping is now a gravity increase so that part is redundant

MoveX: this function shifts the tetromino in the direction specified IT DOES NOT VERIFY IF THE MOVE IS POSSIBLE FIRST. CHECK TRYMOVEX BEFORE CALLING

Softdrop: lowers the active tetromino by 1. Only ever used for the gravity function now ALSO DOES NOT CHECK IF MOVE IS POSSIBLE

LockActive: fills the board in the tetromino's current position then checks for line clears, and game overs, clearing if it can. Also updates highestfillrow

NextPiece: sets the tetromino up to the top of the board with the given monotype

TryRotate: attempts to rotate in the specified direction. First successful srs point is returned.

RotateDir: rotates the tetromino in the given direction with the srs point specified. Relies on tryrotate to get a valid srs point. Also checks for tspins and mini tspins if the tetromino is T

UpdateGhost: updates the ghost piece's position by retesitng it. Called after movement rotation lockdown and holding.

HardDrop: brings the active tetromino onto the ground and immediately locks it in place.

HoldTet: swaps the active tetromino with the hold or the next in the randomizer if hold is empty.

5: DrawingStuff.cs

5.1: General Overview:

This class consists entirely of static variables and functions. Handles updating almost every visual element of the game other than the pause menu and the next queue

5.2: Variables:

Rectangle[20][10] boardDrawArr: similar to GameLogic.board but instead it contains the rectangles used to represent the board.

Rectange[4] activeDrawArr: array of rectangle objects used to draw the active tetromino.
Rectangle[4] ghostDrawArr: same as activeDrawArr but for the ghost piece
Brush[8] brusharr: array of colors used for each tetromino sorted by TetrominoID (3.2)
Image[5] nextdrawarr: array for the next queue view. Called from randomizer.updateDraw
Uri[8] urishortcut: used make next queue and hold piece drawing less cluttered.
Bool maderects: dont recreate the rectangles if they were already made

5.3: Functions:

initialRects: used to create the various rectangles at the start of the game
Updateboard: refreshes the drawing of the board from the highest fill row to the bottom
Drawghost: updates ghostpiece drawing
Updatescores: updates the score level and lines cleared text

6: Randomizer.cs

6.1: General overview:

Generates random pieces according to the 7 bag system as well as displays the next queue

7: MainWindow.xaml.cs

7.1:General Overview:

This class goes along with MainWindow.xaml. Mostly used for input and output. Much of the class is auto generated. Despite its length most of the functions are rather self explanatory.