Lum Fazliu

Programming 2 Semester Project

TETRIS

INDEX

User Manual:

Overview:

Tetris is a classic and addictive puzzle game that challenges players' spatial reasoning and quick decision-making skills. The objective of the game is to manipulate a series of geometric shapes, known as tetrominoes, as they descend from the top of the playing field. Players must rotate and move these tetrominoes to create complete horizontal lines without any gaps. When a line is completed, it disappears, and the player earns points. The game becomes progressively faster and more challenging as players clear lines, requiring them to think and act swiftly to keep the game going. The goal is to survive as long as possible by clearing lines and avoiding letting the stack of tetrominoes reach the top of the screen, at which point the game ends. Tetris is known for its simple yet engaging gameplay that has captivated players of all ages for decades.

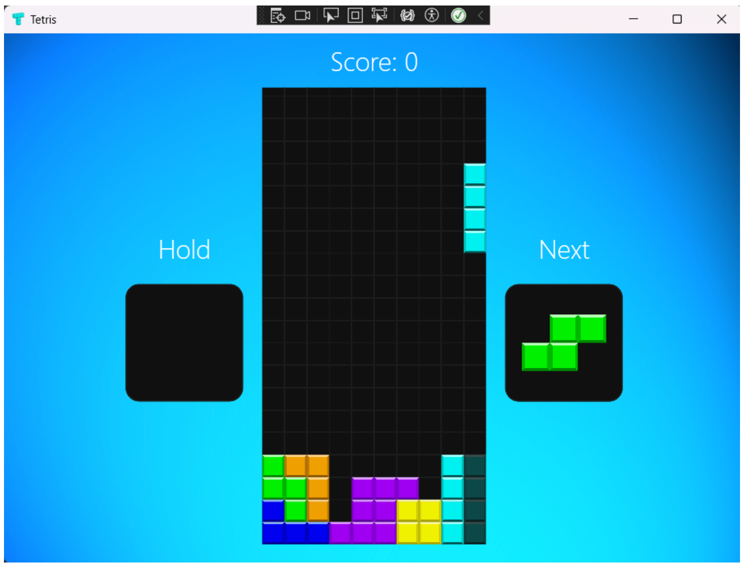
How To Score Higher:

Scoring higher in Tetris involves a combination of strategic planning and quick reflexes.

1. One key strategy is to focus on creating and clearing multiple lines simultaneously, which earns bonus points. To do this, look for opportunities to position tetrominoes in a way that you can clear two or more lines at once, commonly referred to as "Tetris."
2. Additionally, try to keep the playing field as flat as possible, avoiding uneven stacks that are harder to clear. It's crucial to plan ahead and anticipate the best placement for each incoming piece, making use of the hold feature when necessary.
3. Maintaining a clear well (an open vertical space) near the center of the board can be beneficial for accommodating different tetromino shapes.
4. Lastly, staying calm under pressure and making quick decisions is essential as the game speeds up. Consistency and practice will improve your Tetris skills over time, enabling you to achieve higher scores and potentially master this classic puzzle game.

User Interface

This project was created through WPF (Windows Presentation Foundation). It involved leveraging the power of XAML (Extensible Application Markup Language) for designing the user interface and C# for handling the game's logic. I used XAML to create grids for the game board, buttons for controls, text elements for scoring, and instructions on the buttons. The beauty of WPF lies in its flexibility and rich graphical capabilities, enabling the creation of visually appealing Tetris game interfaces with ease. Behind the scenes, C# code handles user input, tetromino movement, when a row becomes full, scoring, and game progression. The integration of XAML and C# in WPF provides a seamless way to build and update the Tetris game interface while ensuring smooth gameplay and responsive user interaction.



The user will use the arrow keys to move the tetromino left, right, down, or change its shape with the upper arrow. As a row becomes full of tetrominoes, the row gets cleared, and the score increments with the number of rows being cleared. The goal is to get the score as high as possible, and of course, it is almost impossible to go forever in this game. My highest score was 30, as I cleared 30 rows.

Decomposition of the Project:

The Project is decomposed into the following directories:

1. *Pictures*
    - *Includes the background picture and the tetrominoes.*
2. *GridLayout*
    - *Creating the Tetris grid. Will describe it more in detail.*
3. *Block*
    - *Creating the block class and setting the next block of the game. Will describe it more in detail.*
4. *Separate Rotation Classes*
    - *Rotating the tetrominoes. Will describe it more in detail.*
5. *Coordinate*
    - *Positioning the tetrominoes. Will describe it more in detail.*
6. *BoardPhase*
    - *Checking the phase of the round in the grid. Will describe it more in detail.*

Software Specifications of Each Directory

GridLayout:

This class is responsible for managing the game board, checking the state of cells within the grid, and clearing full rows when necessary.

Properties

"int Rows"

- Description: Gets the number of rows in the game grid.

"int Columns"

- Description: Gets the number of columns in the game grid.

Indexer

"int this[int r, int c]"

- Description: Allows access to individual cells within the game grid using row and column indices.
- Parameters:
  - "r" (int): The row index.
  - "c" (int): The column index.
- Returns: The value at the specified grid cell.

Constructor

"BoardPhase(int rows, int columns)"

- Description: Initializes a new instance of the GameGrid class with the specified number of rows and columns.
- Parameters:
  - "rows" (int): The number of rows in the grid.
  - "columns" (int): The number of columns in the grid.

Methods

"bool IsInside(int r, int c)"

- Description: Checks if a given row and column pair is inside the boundaries of the game grid.
- Parameters:
  - r (int): The row index to check.

    - c (int): The column index to check.
- Returns: true if the specified cell is within the grid; otherwise, false.

“bool IsEmpty(int r, int c)”

- Description: Checks if a specific cell in the game grid is empty (contains a value of 0) and is within the grid boundaries.
- Parameters:
    - r (int): The row index of the cell.
    - c (int): The column index of the cell.
- Returns: true if the cell is empty and within the grid; otherwise, false.

“bool IsRowFull(int r)”

- Description: Checks if an entire row in the game grid is full (contains no empty cells).
- Parameters:
    - r (int): The row index to check.
- Returns: true if all cells in the row are filled; otherwise, false.

“bool IsRowEmpty(int r)”

- Description: Checks if an entire row in the game grid is empty (contains only empty cells).
- Parameters:
    - r (int): The row index to check.
- Returns: true if all cells in the row are empty; otherwise, false.

“int RemoveCompleteRow()”

- Description: Clears all full rows in the game grid and shifts the rows above down to fill the empty space.
- Returns: The number of rows cleared in the operation.
- Note: This method combines the previous methods to clear full rows efficiently.

Block:
The Block class in the Tetris namespace serves as an abstract base class for defining different Tetris blocks. These blocks are the fundamental shapes that players manipulate in the game. This class provides essential properties and methods for managing block states, positions, and rotations within the game.

Properties

“protected abstract Position[][] Tiles”

- Description: Represents the array of positions that define the shape of the block. Each position is relative to the block's anchor point.

“protected abstract Position StartOffset”

- Description: Defines the starting offset position for the block. This offset determines the initial position of the block on the game grid.

“public abstract int Id”

- Description: Provides a unique identifier for the block type. This identifier is used for distinguishing different block shapes.

Fields

“private int rotationState”

- Description: Stores the current rotation state of the block.

“private Position offset”

- Description: Represents the offset position of the block on the game grid.

Constructor

“Block()”

- Description: Initializes a new instance of the Block class, setting the initial offset to the specified start offset.

Methods

“IEnumerable<Coordinate> TilePositions()”

- Description: Generates a sequence of positions for all tiles (cells) occupied by the block, taking into account the current rotation state and offset.
- Returns: An enumerable collection of positions.

“void Move(int rows, int columns)”

- Description: Moves the block by a specified number of rows and columns on the game grid, updating its offset position.

"void Reset()"

- Description: Resets the block to its initial state, including setting the rotation state and offset back to their original values.

The BlockQueue class in the Tetris namespace manages the sequence of Tetris blocks that will appear in the game. It handles the random selection of blocks and ensures that the same block does not appear consecutively.

Fields

private readonly Block[] blocks

- Description: An array containing all possible Tetris blocks.

private readonly Random random

- Description: A random number generator used to select blocks from the array.

Properties

public Block NextBlock

- Description: Gets the next Tetris block that will appear in the game.

Constructors

BlockQueue()

- Description: Initializes a new instance of the BlockQueue class. It sets the NextBlock property to a randomly selected initial block.

Methods

private Block RandomBlock()

- Description: Selects a random Tetris block from the array of available blocks.
- Returns: A randomly selected Tetris block.

public Block SetAndGenerate()

- Description: Retrieves the current NextBlock, updates NextBlock to a new random block (ensuring it is not the same as the current block), and returns the current block.
- Returns: The current NextBlock.

Coordinate:
The Coordinate class in the Tetris namespace represents a simple data structure used to store the row and column coordinates of a point within the game grid. This class is commonly used throughout the Tetris game to define the positions of blocks, tiles, and other game elements.

Properties

“public int Row”

- Description: Gets or sets the row coordinate of the position.

“public int Column”

- Description: Gets or sets the column coordinate of the position.

Constructor

“Position(int row, int column)”

- Description: Initializes a new instance of the Coordinate class with the specified row and column coordinates.
- Parameters:
    - row (int): The row coordinate.
    - column (int): The column coordinate.

BoardPhase:
The GameState class in the Tetris namespace manages the state and logic of a Tetris game. It tracks the current block, game grid, scoring, and various game interactions such as block movements, rotations, and holding.

Properties

“public Block CurrentBlock”

- Description: Gets or sets the currently active Tetris block.
- Notes: The setter resets the block's position and checks if it fits in the grid's initial position.

“public GameGrid GridLayout”

- Description: Gets the game grid that represents the playing area.

“public BlockQueue BlockQueue”

- Description: Gets the block queue responsible for generating new blocks.

“public bool RoundOver”

- Description: Gets a value indicating whether the game is over.

“public int Score”

- Description: Gets the current player's score.

“public Block HeldBlock”

- Description: Gets or sets the block that is currently being held by the player.

“public bool CanHold”

- Description: Gets a value indicating whether the player can hold the current block.

Constructors

BoardPhase()

- Description: Initializes a new instance of the BoardPhase class, creating a game grid and block queue. It sets up the initial game state.

Methods

public void HoldBlock()

- Description: Allows the player to swap the current block with the held block if the hold feature is available.

public void RotateBlockCW()

- Description: Rotates the current block 90 degrees clockwise (CW) if it fits within the grid.

public void RotateBlockCCW()

- Description: Rotates the current block 90 degrees counterclockwise (CCW) if it fits within the grid.

public void MoveBlockLeft()

- Description: Moves the current block one column to the left if it fits within the grid.

public void MoveBlockRight()

- Description: Moves the current block one column to the right if it fits within the grid.

public void MoveBlockDown()

- Description: Moves the current block one row down. If the block cannot move further down, it places the block on the grid and checks for completed rows.

public int BlockDropDistance()

- Description: Calculates the maximum distance the current block can drop within the grid before hitting another block or the bottom.

public void DropBlock()

- Description: Instantly drops the current block to its maximum possible position within the grid and places it. This method is called when the player chooses to drop the block.

Private Methods

private bool BlockFits()

- Description: Checks if the current block fits within the grid without colliding with other blocks.

private bool IsGameOver()

- Description: Determines if the game is over by checking if the top rows of the grid are occupied.

private void PlaceBlock()

- Description: Places the current block on the grid, updating the grid's state and checking for completed rows. If the game is not over, it retrieves the next block from the block queue and allows for block holding.

private int TileDropDistance(Position p)

- Description: Calculates the vertical distance a single tile in the current block can drop before hitting another block or the bottom.