**CUSTOM PROGRAM DETAILED PLAN**

**PART A**

**I. Student Information**

- Name: Truong Ngoc Gia Hieu

- Student ID (HCMC): SWS01217

- Student ID (Australia ID): 105565520

**PART B**

**2.1. Custom Program Title:** Tetris Game

**2.2. Overview of Custom Program**

The fascinating project Tetris game implementation showcases a digital version of the classic puzzle game. Moreover, the program challenge players to rotate and strategically place falling blocks (tetrominoes) to create complete horizontal lines, which then disappear, earning points. Furthermore, throughout the level, the game increases in speed and complexity as the player progresses, providing a continuous challenge. Notably, the game illustrates the understanding of four principles concepts of OOP including Abstraction, Encapsulation, Inheritance, and Polymorphism.

**2.3. Project Description Overvall**

The game Tetris (Block game) consists of a grid include 22 rows (2 hidden rows) and 10 colunns. To achieve the D level, the game Tetris implements principles of Object-Oriented Programming (OOP). The gird will be in the center with black cells and white lines between cells. About the logic game, when start the game, random blocks will drop down and player can utilize arrow keys on the keyboard to move blocks to left, right, and drop down faster. By the way, column on the left and right side will also work. Interestingly, the normal logic game of the Tetris will be normal.

Moreover, on the left side, there will have test “Next” and under that word, a black rounded-corner rectangle will contains next random blocks. Notably, the game will have special function which allows the player to store the block if they want by pressing “S” (Store) key in the keyboard. As a result, the stored block and text “Store”will be put below the “Next”. Similarly, on the right side of the grid, the text “Current” will display and current block will be displayed in the black rounded-corner rectangle that is below the text. Furthermore, the Score will be also implemented too.

In summary, this is my initial plan for the custom program Tetris game which achieve level.

**2.4. Classes, Purposes & UML Design**

A screenshot of a computer program

AI-generated content may be incorrect.**2.4.1 GameGrid class**

- The ***GameGrid*** class holds two-dimensional rectangular array that represents the game’s grid where different game pieces (Tetris blocks) will be placed. Therefore, the class includes following attributes, constructor, and methods as the UML design showcase.

+ ***this [int r, int c]:*** The indexer provides easy access to the array including fetches the value located at the specified row (r) and column (c) from grid and sets the value at the specified row and column in the grid.

+ ***IsInside():*** Method for checks that inside the grid or not

+ ***IsEmpty():*** Method for checking if a given cell is empty or not

+ ***IsRowFull():*** Method for checking if a row is empty or not.

***+ IsRowEmpty():*** Method for checking if a row is empty or not

***+ ClearFullRows():*** Clear full rows

***-*** Two methods ***ClearRow()*** and ***MoveRowDown()*** methods are set privately due to several reasons:

***+* Encapsulation (Control Access):** By setting two methods private, the class ensuring that they are not accessible from outside the ‘***GameGrid’*** class. As a result, this restricts their use to within the class itself, which helps maintain the integrity of the class's state.

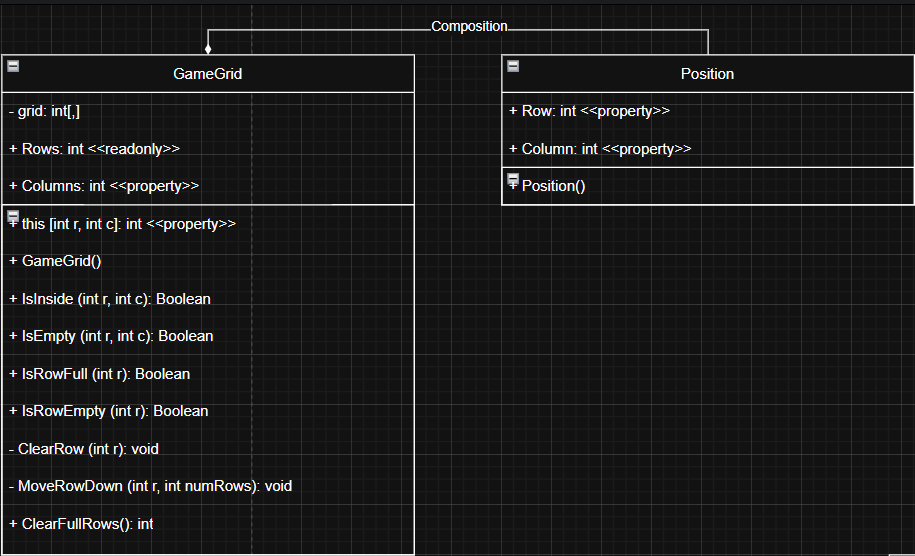
**+ Intermal Functionality:** ‘***ClearRow’*** and ‘***MoveRowDown’*** are utility methods that are used for managing the internal mechanics of the game grid. Other classes do not need to directly modify how rows are cleared or moved; instead, they should interact with public methods like ClearFullRows() that manage these processes.

**2.4.2 Position class**

A black grid with white text

AI-generated content may be incorrect.- UML Design

- Relationship with the GameGrid class:



- While the GameGrid class defines the overall structure of game development, Position class represents specific locations within that environment. Consequently, **GameGrid** likely contains or uses instances of **Position** to manage and represent the grid's layout and movement.

**2.4.3 Block class**

A screen shot of a computer

AI-generated content may be incorrect.

- The method ***TilePositions()*** using ***IEnumerable<Position>*** loops over the tile positions in the current rotation state and adds the row offset and column offset.

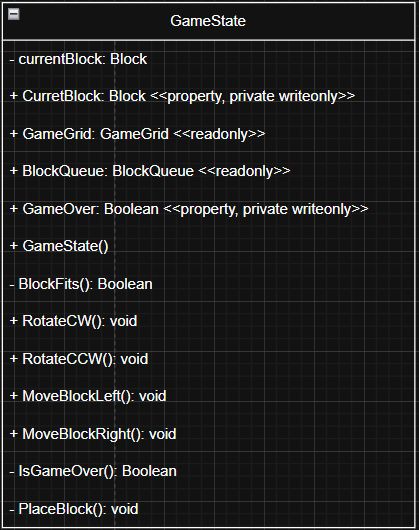
- The ***RotateCW()*** method designs to rotate a Tetris block in a clockwise direction. For more details, it updates the **rotationState** variable, which keeps track of the current orientation of the block. Particularly, tshe expression ‘**(rotationState + 1) % Tiles.Length’** ensures that when the maximum number of rotations (defined by the length of **Tiles**) is reached, it wraps around back to the first rotation, allowing for continuous rotation without exceeding the bounds of the array.

- The ***RotateCCW()*** rotates a Tetris block in a counter-clockwise direction. Specificially, method first checks if the **rotationState** is at its initial position (0). If it is, it sets the **rotationState** to the last index of **Tiles** (i.e., **Tiles.Length - 1**), thus wrapping around.If the **rotationState** is not at 0, it simply decrements the **rotationState** by one, allowing the block to rotate to the previous orientation.

A screen shot of a computer

AI-generated content may be incorrect.**2.4.4 Block classes (Iblock, Jblock, TBlock, ZBlock, Oblock, Sblock, and Lblock.cs)**

**2.4.5 GameState class**



The ***GameState*** class is extremely important because It handles interactions between previous classes in the Tetris game, facilitating gameplay through various methods as the UML Desgin above. Additionally, the class implements a grid of 22 rows and 10 columns and updating the current block from the queue. Notably, the class contains methods to check if the current block fits within the grid, rotate it both clockwise and counter-clockwise, and move it left or right. Then, it monitors the game state with functionalities to determine if the game is over and to place the block into the grid, clearing full rows as needed. Consequently, the structured approach enables efficient gameplay while ensuring proper interactions among game components.

**2.4.6 MainWindow.xaml.cs classs**

**2.5. UML Design Class and Relationships:** The relationship between classes in the Tetris solution is showcased on the pdf file named ‘***UML\_Design\_and\_Relationship’***.