Design Overview for SplashTris

Name: Luu Tuan Hoang
Student ID: 104180391

Summary of Program

A game implementation based on the well-known puzzle game Tetris is the SplashTris program. The SplashKit SDK, which offers a graphical user interface and a game framework for development, is intended to be used to run the program.

The program's main feature is the creation of a grid-based game board on which tetromino shapes, which are made up of four square blocks, descend. As these shapes fall, the player's task is to move and rotate them in order to arrange them into full horizontal lines. A line vanishes after it is finished, and the player gains points.

The implementation makes use of input functions to manage player interactions via keyboard controls and rendering functions of SplashKit to display the game elements on the screen. The logic of the game also includes line clearing, shape manipulation, collision detection, and scoring calculations.

Players can start and restart the game as they try to improve their Tetris skills and earn high scores. The program faithfully recreates the original Tetris gameplay while offering an enjoyable and challenging gaming experience.

Required Roles

Responsibility	Type Details
GameState	The primary game logic and control flow are represented by this class. It controls game state variables like the level, score, and lines cleared. It manages user input, updates game components (like the tetrominoes that are falling), and carries out collision detection and line clearing.
Tetromino	A tetromino shape is represented by this abstract class. It keeps track of the shape's position, rotation state, and individual building blocks. It offers ways to move and rotate the shape.
Position	It provides methods for checking cell occupancy and performing operations on the grid, such as clearing completed lines.
Grid	Game grid of gameboard. Keeps track of the state of each cell in the grid
Queue	Manage the tetromino queue with bag randomization technique
UI	Draw the interface for the game using SplashkitSDK
DrawHighScore	Manage high scores and sort the score in descending order.
Program	Entry point of the game with input controller

Added complexity

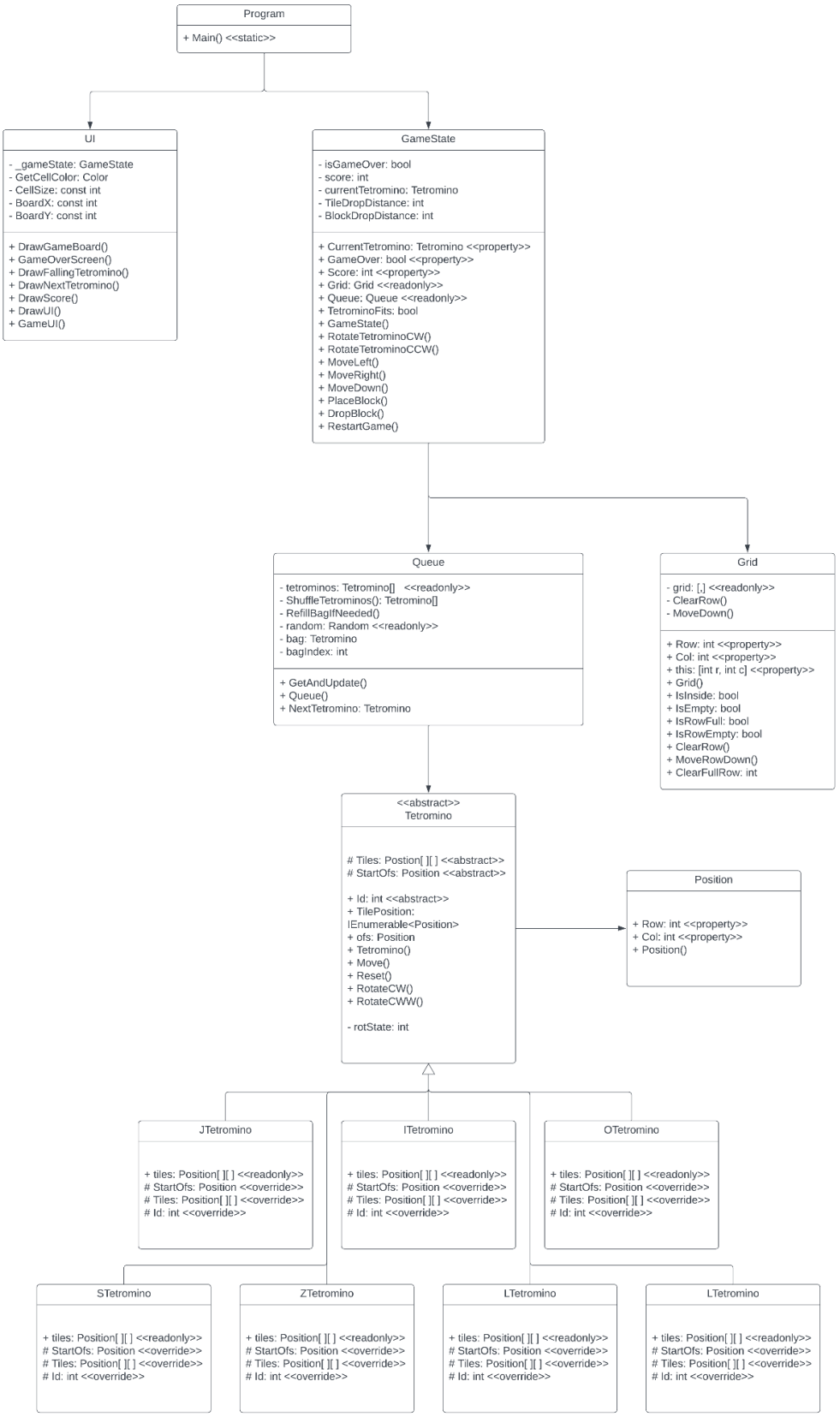
The following complexities are added into the program:

- Leaderboard: Reads and writes score in a .txt file, takes five highest scores in descending order and displays it in the leaderboard section of the UI.
- Bag randomization technique: The Tetris bag randomization technique creates a bag containing all seven tetromino shapes, shuffles them at random, and then sequentially gives each shape to the player to ensure a fair distribution of the various tetromino shapes. Each shape moves to the bottom of the bag after it is used, preventing repeated sequences. This strategy ensures that each shape only appears once during each cycle through the bag, preserving a balanced gaming experience and preventing tiresome repetitions of shapes.
- Aesthetic: The game UI uses straightforward drawing commands to create and update the tetromino shapes dynamically rather than loading bitmap images for each one and rotating them while the game is being played. By handling the rendering process programmatically and effectively with the draw method, the game can more easily manipulate the game elements in real-time and offer a fluid and responsive user experience. Additionally, by using this method, fewer image assets are required, which makes the game lighter and easier to manage in terms of storage and performance.
- The speed at which the tetrominoes descend is dynamically changed in the Tetris game according to how well the player is doing. The level rises as the game goes on, causing the tetromino drops to happen more quickly. This is accomplished by gradually decreasing the moveDownInterval, which makes the game harder. The interval between difficulty increments is determined by the timeToIncreaseDifficulty. When you reach the maximumLevel, the difficulty doesn't change. The game also lets players restart after the game is over and records and displays their scores. Color of tetrominoes also changes after each level.

Design pattern

- Singleton Pattern: The Singleton pattern is used when you need to ensure that there is only one instance of a class throughout the application. In the program, multiple instances of classes like GameState and GameUI enforce a single instance for these classes within the Main method.
- Abstract class Tetromino, which represents a base class for different types of Tetrominoes used in the Tetris game. The class includes methods to rotate, move, and reset the Tetromino, as well as properties for representing the Tetromino's different tile positions, starting offset, and unique ID.
- This pattern is used to change an object's behavior when its internal state changes. In the given code, the game's behavior is handled primarily in the GameState class.

UML Diagram


Program
+ Main() <<static>>
UI
- _gameState: GameState
- GetCellColor: Color
- CellSize: const int
- BoardX: const int
- BoardY: const int
+ DrawGameBoard()
+ GameOverScreen()
+ DrawFallingTetromino()
+ DrawNextTetromino()
+ DrawScore()
+ DrawUI()
+ GameUI()
GameState
- isGameOver: bool
- score: int
- currentTetromino: Tetromino
- TileDropDistance: int
- BlockDropDistance: int
+ CurrentTetromino: Tetromino <<property>>
+ GameOver: bool <<property>>
+ Score: int <<property>>
+ Grid: Grid <<readonly>>
+ Queue: Queue <<readonly>>
+ TetrominoFits: bool
+ GameState()
+ RotateTetrominoCW()
+ RotateTetrominoCCW()
+ MoveLeft()
+ MoveRight()
+ MoveDown()
+ PlaceBlock()
+ DropBlock()
+ RestartGame()
Queue
- tetrominos: Tetromino[] <<readonly>>
- ShuffleTetrominos(): Tetromino[]
- RefillBagIfNeeded()
- random: Random <<readonly>>
- bag: Tetromino
- bagIndex: int
+ GetAndUpdate()
+ Queue()
+ NextTetromino: Tetromino
Grid
- grid: [,] <<readonly>>
- ClearRow()
- MoveDown()
+ Row: int <<property>>
+ Col: int <<property>>
+ this: [int r, int c] <<property>>
+ Grid()
+ IsInside: bool
+ IsEmpty: bool
+ IsRowFull: bool
+ IsRowEmpty: bool
+ ClearRow()
+ MoveRowDown()
+ ClearFullRow: int
<<abstract>>
Tetromino
Tiles: Postion[ ][ ] <<abstract>>
StartOfs: Position <<abstract>>
+ Id: int <<abstract>>
+ TilePosition:
IEnumerable<Position>
+ ofs: Position
+ Tetromino()
+ Move()
+ Reset()
+ RotateCW()
+ RotateCWW()
- rotState: int
Position
+ Row: int <<property>>
+ Col: int <<property>>
+ Position()
JTetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
ITetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
OTetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
STetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
ZTetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
LTetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>
LTetromino
+ tiles: Position[ ][ ] <<readonly>>
# StartOfs: Position <<override>>
# Tiles: Position[ ][ ] <<override>>
# Id: int <<override>>

Advancement

The new program design exhibits improvements in terms of code organization, abstraction, reusability, and maintainability. It demonstrates a higher level of complexity and follows best practices, making it a more well-done design compared to a D-level custom program design. It makes use of many libraries that require complex understanding like System.Linq, System.Collections.Generic, System.Diagnostics, System.IO and SplashKitSDK.