Design Overview for Gibbous Tetris

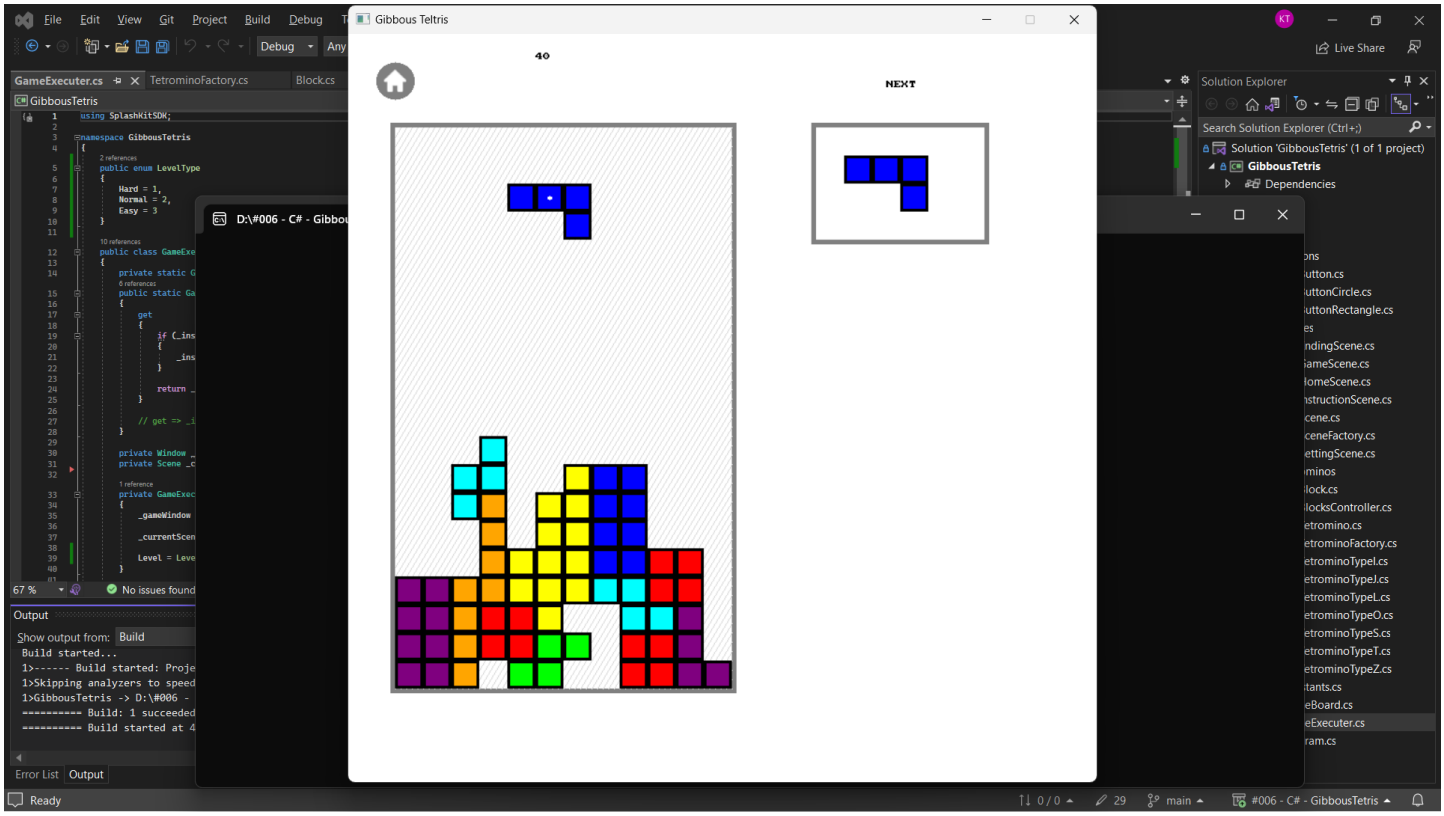
Name: Trung Kien Nguyen
Student ID: 104053642

Summary of Program

Describe what you want the program to do… one or two paragraphs. Include a sketch of sample output to illustrate your idea.

This program will basically provide a grid-based playing field where various types of Tetrimino, which is a shape composed of four blocks (squares), fall from the top of the screen. The player's objective is to move and rotate the falling Tetriminos to create complete horizontal lines without any gaps. When a line is completed, it disappears, and the player earns points. As the game progresses, the playing field is gradually filled, challenging the player's reflexes and strategic thinking. The game continues until the stacked Tetriminos reach the top of the grid, at which point the game ends. The program should display the player's current score and provide an option to restart the game for continuous play.

In addition, the program has a total of 5-6 scenes, including the home scene (for choosing the difficulty level) main gameplay scene, instruction scene (for how-to-play), setting the scene (for advanced settings like the sound manager), ending scene for the result, and replaying, and also an intro scene (if possible).



Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Table 1: <<role name>> details – duplicate

Game Executer

Responsibility	Type Details	Notes
Instance	GameExecuter	Singleton
Execute()	void	Loop for execute the Splashkit program
Update()	void	For all related objects to Update
Draw()	void	For all related objects to Draw
ChangeScene()	void	Change between scenes

Scene

Responsibility	Type Details	Notes
Update()	void	For all related objects to Update
Draw()	void	For all related objects to Draw

Block

Responsibility	Type Details	Notes
X, Y	double	Position in screen
Xindex, Yindex	double	Position in the 2D-array
Draw()	void	For all related objects to Draw

Tetromino: Has 4 blocks each

Responsibility	Type Details	Notes
TheTetromino	List<Block>	4 blocks in a tetromino
Update()	void	For all related objects to Update
Draw()	void	For all related objects to Draw
Rotate()	void	
MoveLeft()	void	
MoveRight()	void	
MoveDown()	void	
MoveToBottom()	void	Move until there are blocks below

BlocksController: Control all the blocks that have been terminated as a 2D-array

Responsibility	Type Details	Notes
Instance	BlocksController	Singleton
Update()	void	For all related objects to Update
Draw()	void	For all related objects to Draw
BlocksIndex	int[,]	List of terminated blocks' position (in indexes)
BlocksDisplay	List<Block>	List of terminated blocks

TetrominoFactory: Factory Method Pattern, to generate Tetromino

Responsibility	Type Details	Notes
CreateTetromino()	Tetromino	

Table 2: <<enumeration name>> details

(Tetromino.)Angle: for the current angle/direction of the tetrominos

Value	Notes
Up	
Right	

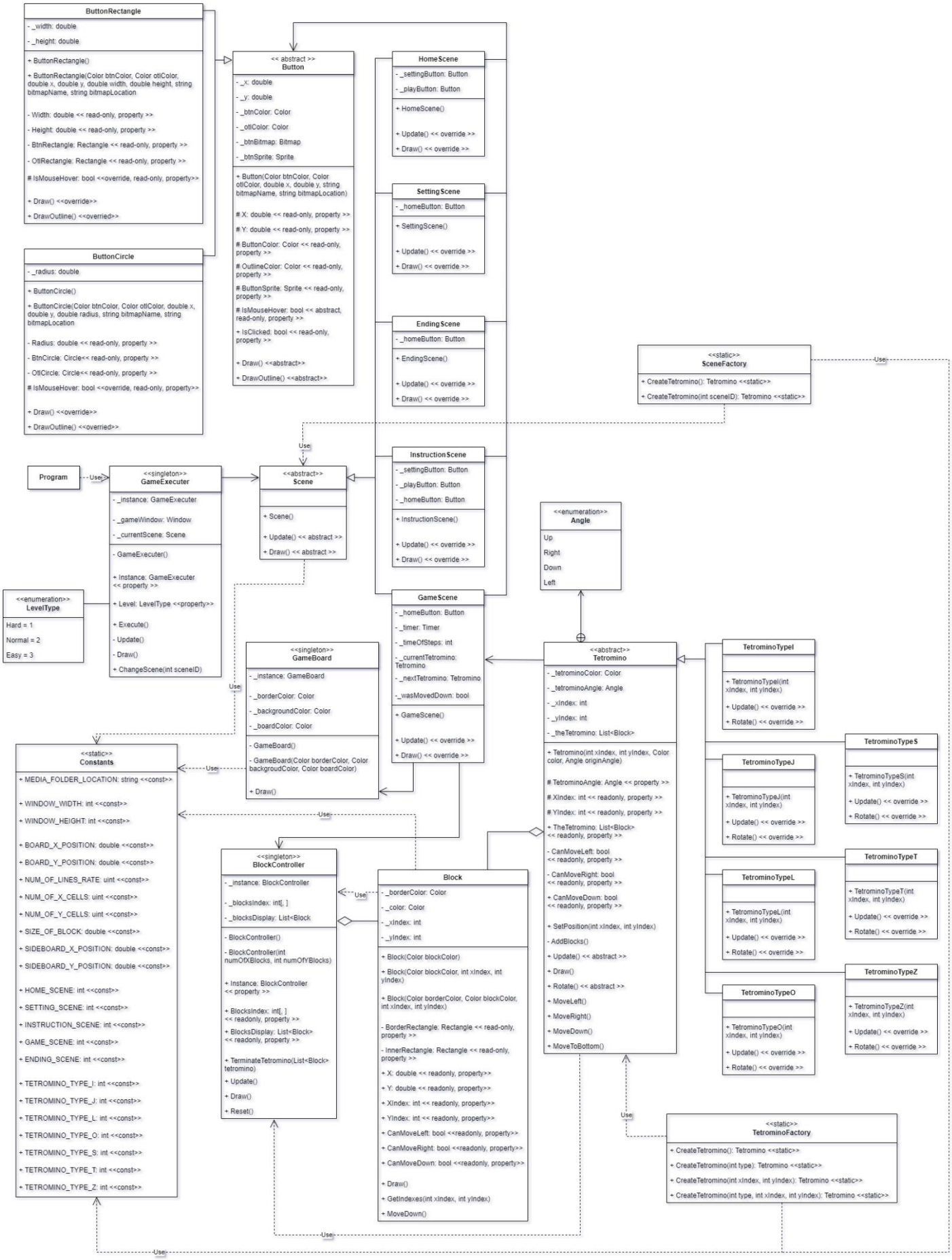
Left	
Down	

LevelType: difficulty of the game (index represents the amount of time for each drop of the tetromino)

Value	Notes
Hard	= 1 (second)
Normal	= 2
Easy	= 3

Class Diagram

Provide an initial design for your program in the form of a class diagram.



Sequence Diagram

Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.

HD criteria

1. Design patterns I may use:
   - Factory Method Pattern: This is a creational design pattern that provides a factory class for creating objects but allows subclasses to decide which class to instantiate. In my GibbousTetris, each type of Tetrimino is represented by a specific object created from a class that inherits from the abstract class “Tetromino”.
     - Create Tetromino randomly: By using this design pattern, I can introduce randomness in the creation process of Tetriminos. The factory class can generate a random number to determine which type of Tetrimino to create, providing a varied and hard-to-predict gameplay experience.
     - Ensure Abstraction and Encapsulation: It encapsulates the creation logic within the factory method or factory class, which is separated from the rest of the codebase.
     - Apply Polymorphism: Obviously, each subclass of Tetromino can have its own implementation of the factory method, enabling the creation of different types of Tetriminos based on the specific rules and characteristics
     - Flexibility: When there are new types of tetromino added to the game in the future added to the game in the future, I can easily extend the factory class and create new subclasses of Tetromino without modifying the existing code in the Main.
     - Besides the tetrominos, there is a total of 5-6 scenes and they may also need a specifc factory class to create, it is better to apply Factory Method Pattern too.
   - Singleton: There are some classes at which having multiple instances would be undesirable or contradictory to the intended design, including:
     - “BlockController”: to control the blocks that have been terminated as a 2D array
     - “GameExecuter”: Obviously, this is responsible for executing scene so there MUST be only one instance.
     - “GameBoard”: This class is to draw and decorate the game area in the game scene.
   - Observer pattern: I may be used this to provide a way for different components or objects to observe and respond to changes in the game state (update later).