Design Overview for Gibbous Tetris

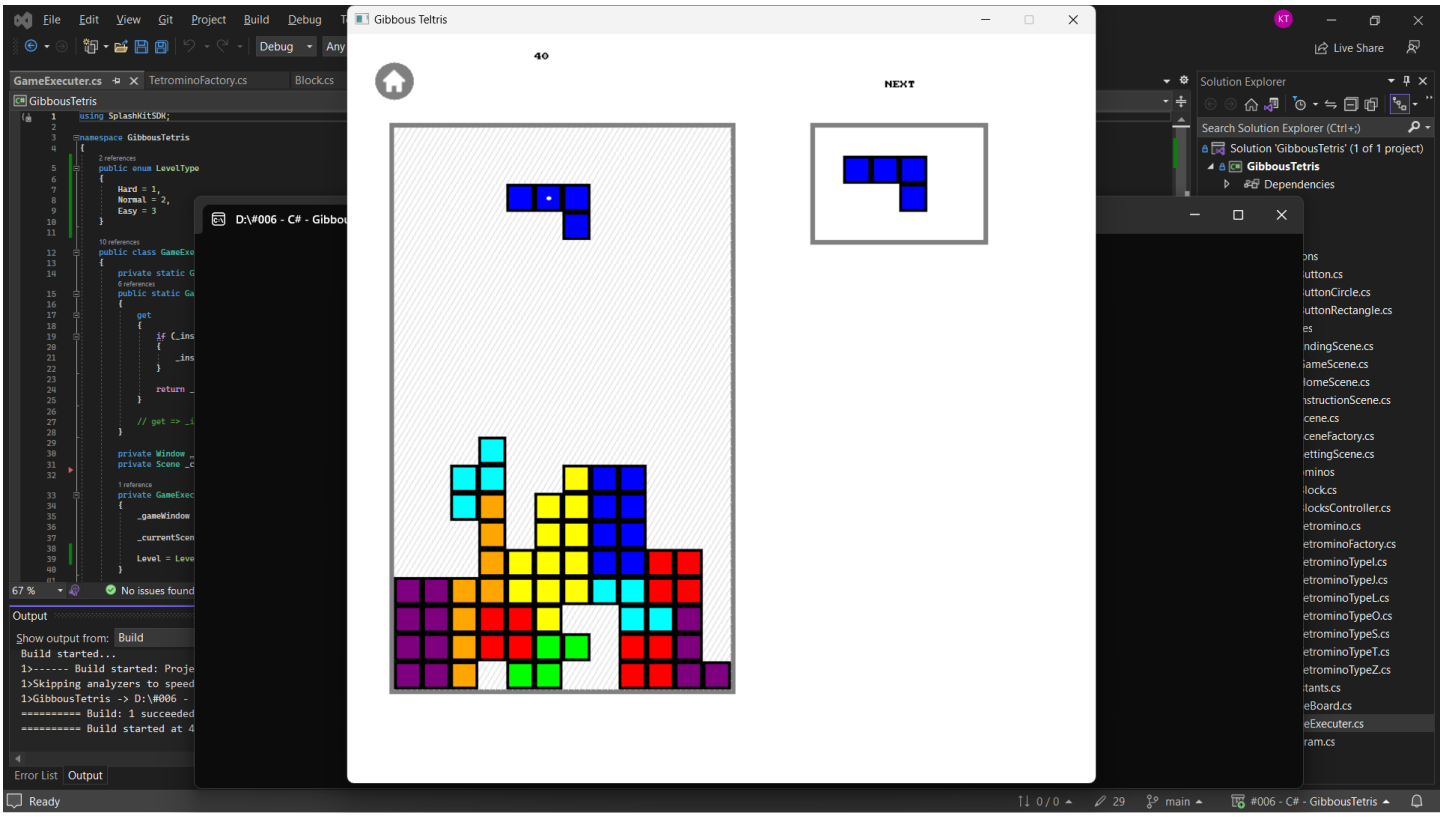
Name: Trung Kien Nguyen
Student ID: 104053642

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.

Tables 1: <<class>> 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

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
CanMoveDown	bool	
CanMoveLeft	Bool	
CanMoveRight	bool	

Tetromino: Has 4 blocks each

Responsibility	Type Details	Notes
_tetrominoColor	(SplashKit.)Color	
_tetrominoAngle	(Tetromino.)Angle	
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
RotateIndexesClockwise	Point2DIndex	Rotate the block's offset clockwise
GetIndexOffsets()	Point2DIndex[]	
TetrominoOffsets()	Point2DIndex[]	Offset of a block is the distance from the center one to it (in indexes)
TetrominoIndexPoints()	Point2DIndex[]	With the offsets of the four blocks, and the ceter point (in indexes), I can calculate the coordinates of those blocks (in indexes)
CheckIfEmpty	bool	if space for blocks are empty in the case of the assumed Angle of the tetromino

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	= 0
Right	= 1
Left	= 2
Down	= 3

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

Table 3: <<struct name>> details

Point2Dindex: Similar as SplashKit.Point2D, but in integers

Value	Notes
Xindex	= 0
YIndex	= 1

Table 4: <<inteface name>> details

IObserver: for implementing simple Observer pattern

Value	Type Details	Notes
OnNotify	void	

Table 5: <<abstract class name>> details

ObservableSubject: for implementing simple Observer pattern

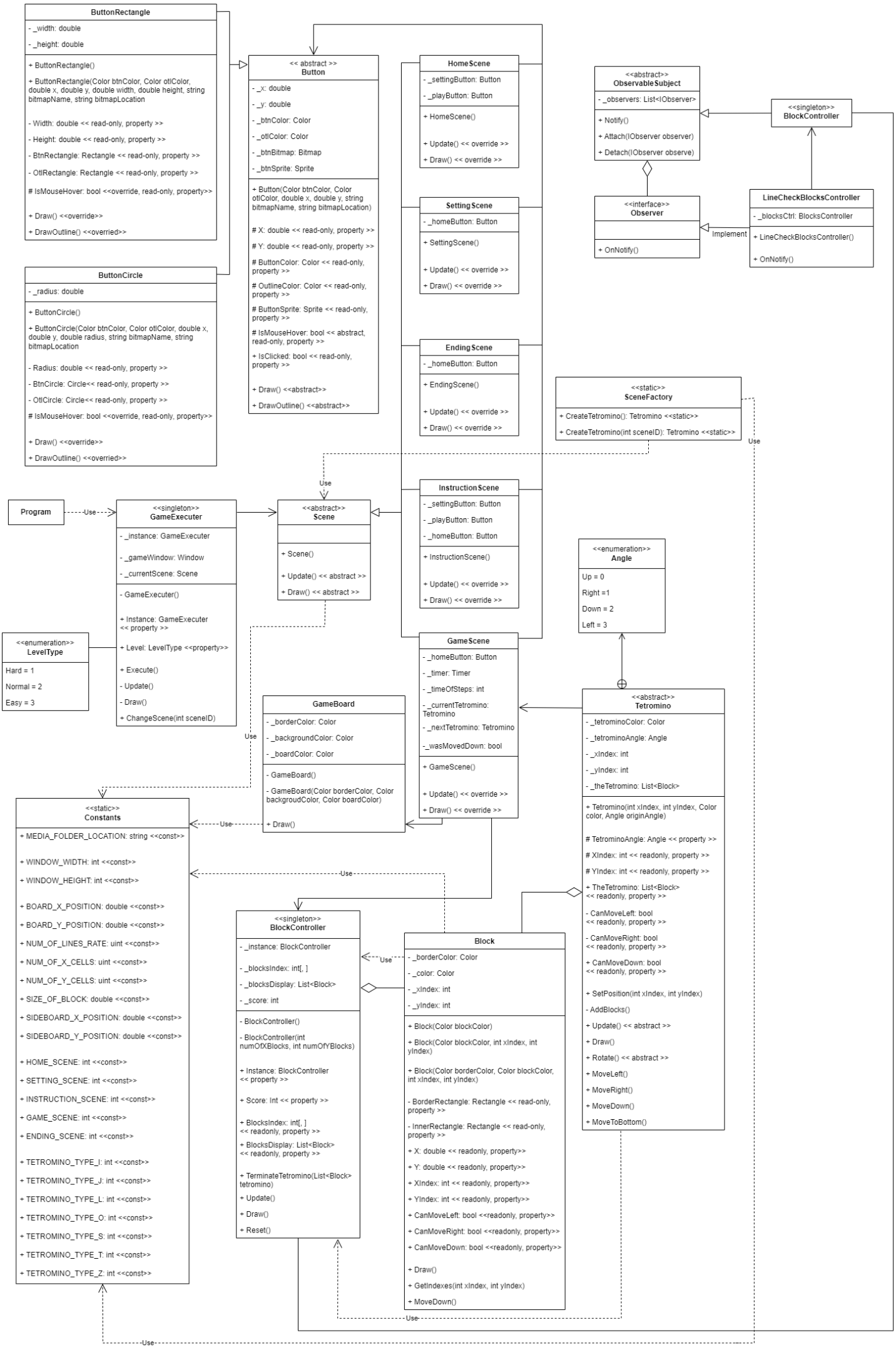
Value	Type Details	Notes
_observers	List<IObserver>	
Notify()	void	
Attach()	void	
Detach()	void	

Scene

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

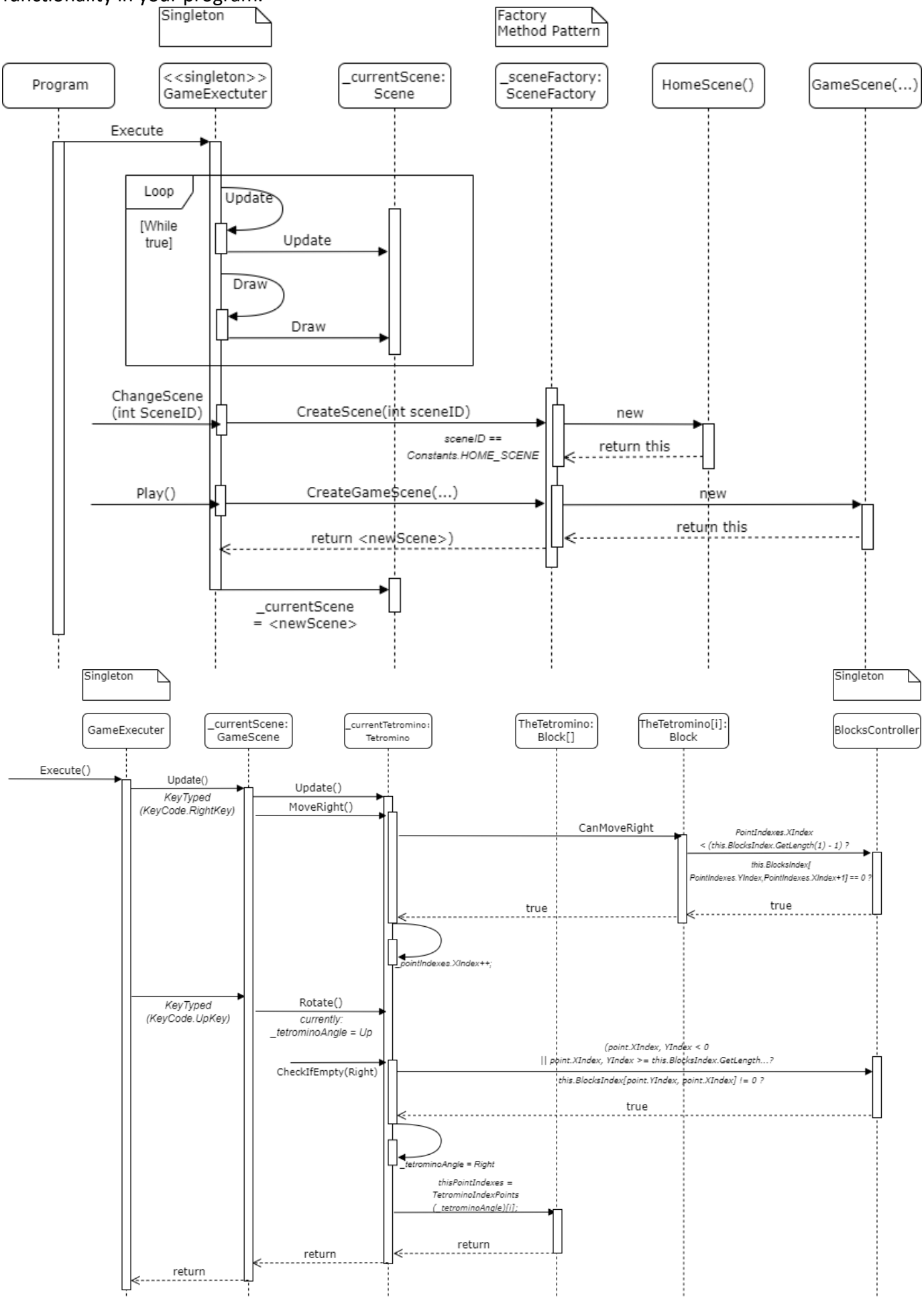
Class Diagram

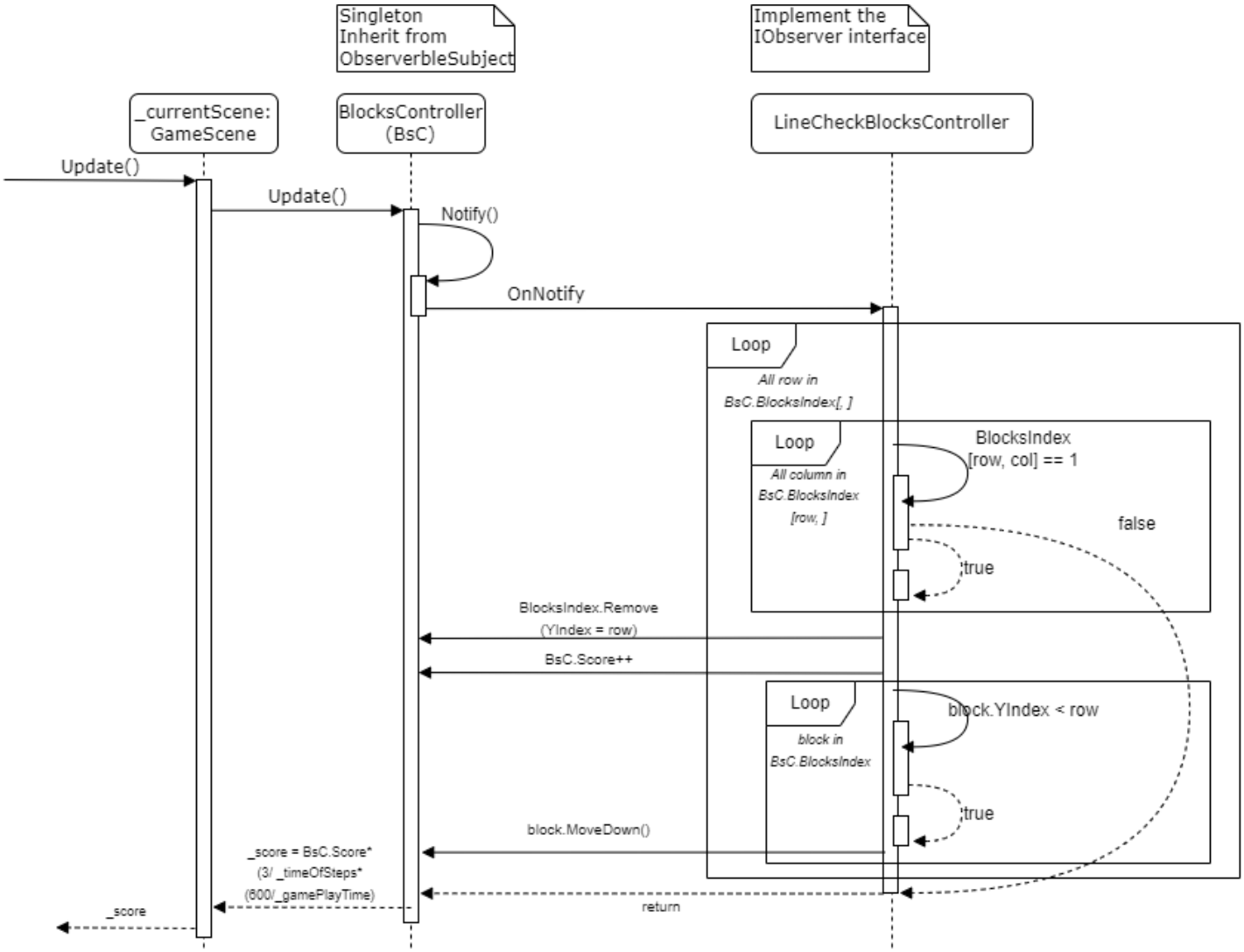
Provide an initial design for your program in the form of a class diagram.


ButtonRectangle
- _width: double
- _height: double
+ ButtonRectangle()
+ ButtonRectangle(Color btnColor, Color otlColor, double x, double y, double width, double height, string bitmapName, string bitmapLocation
- Width: double << read-only, property >>
- Height: double << read-only, property >>
- BtnRectangle: Rectangle << read-only, property >>
- OtlRectangle: Rectangle << read-only, property >>
IsMouseHover: bool <<override, read-only, property>>
+ Draw() <<override>>
+ DrawOutline() <<overried>>
ButtonCircle
- _radius: double
+ ButtonCircle()
+ ButtonCircle(Color btnColor, Color otlColor, double x, double y, double radius, string bitmapName, string bitmapLocation
- Radius: double << read-only, property >>
- BtnCircle: Circle<< read-only, property >>
- OtlCircle: Circle<< read-only, property >>
IsMouseHover: bool <<override, read-only, property>>
+ Draw() <<override>>
+ DrawOutline() <<overried>>
<< abstract >>
Button
- _x: double
- _y: double
- _btnColor: Color
- _otlColor: Color
- _btnBitmap: Bitmap
- _btnSprite: Sprite
+ Button(Color btnColor, Color otlColor, double x, double y, string bitmapName, string bitmapLocation)
# X: double << read-only, property >>
# Y: double << read-only, property >>
# ButtonColor: Color << read-only, property >>
# OutlineColor: Color << read-only, property >>
# ButtonSprite: Sprite << read-only, property >>
# IsMouseHover: bool << abstract, read-only, property >>
+ IsClicked: bool << read-only, property >>
+ Draw() <<abstract>>
+ DrawOutline() <<abstract>>
HomeScene
- _settingButton: Button
- _playButton: Button
+ HomeScene()
+ Update() << override >>
+ Draw() << override >>
SettingScene
- _homeButton: Button
+ SettingScene()
+ Update() << override >>
+ Draw() << override >>
EndingScene
- _homeButton: Button
+ EndingScene()
+ Update() << override >>
+ Draw() << override >>
InstructionScene
- _settingButton: Button
- _playButton: Button
- _homeButton: Button
+ InstructionScene()
+ Update() << override >>
+ Draw() << override >>
GameScene
- _homeButton: Button
- _timer: Timer
- _timeOfSteps: int
- _currentTetromino: Tetromino
- _nextTetromino: Tetromino
-_wasMovedDown: bool
+ GameScene()
+ Update() << override >>
+ Draw() << override >>
<<abstract>>
ObservableSubject
- _observers: List<IObserver>
+ Notify()
+ Attach(IObserver observer)
+ Detach(IObserver observe)
<<singleton>>
BlockController
<<interface>>
Observer
+ OnNotify()
Implement
LineCheckBlocksController
- _blocksCtrl: BlocksController
+ LineCheckBlocksController()
+ OnNotify()
<<static>>
SceneFactory
+ CreateTetromino(): Tetromino <<static>>
+ CreateTetromino(int sceneID): Tetromino <<static>>
Use
Program
Use
<<singleton>>
GameExecuter
- _instance: GameExecuter
- _gameWindow: Window
- _currentScene: Scene
- GameExecuter()
+ Instance: GameExecuter << property >>
+ Level: LevelType <<property>>
+ Execute()
- Update()
- Draw()
+ ChangeScene(int sceneID)
<<abstract>>
Scene
+ Scene()
+ Update() << abstract >>
+ Draw() << abstract >>
<<enumeration>>
LevelType
Hard = 1
Normal = 2
Easy = 3
<<enumeration>>
Angle
Up = 0
Right =1
Down = 2
Left = 3
<<abstract>>
Tetromino
- _tetrominoColor: Color
- _tetrominoAngle: Angle
- _xIndex: int
- _yIndex: int
- _theTetromino: List<Block>
+ Tetromino(int xIndex, int yIndex, Color color, Angle originAngle)
# TetrominoAngle: Angle << property >>
# XIndex: int << readonly, property >>
# YIndex: int << readonly, property >>
+ TheTetromino: List<Block> << readonly, property >>
- CanMoveLeft: bool << readonly, property >>
- CanMoveRight: bool << readonly, property >>
+ CanMoveDown: bool << readonly, property >>
+ SetPosition(int xIndex, int yIndex)
- AddBlocks()
+ Update() << abstract >>
+ Draw()
+ Rotate() << abstract >>
+ MoveLeft()
+ MoveRight()
+ MoveDown()
+ MoveToBottom()
GameBoard
- _borderColor: Color
- _backgroundColor: Color
- _boardColor: Color
- GameBoard()
- GameBoard(Color borderColor, Color backgroudColor, Color boardColor)
+ Draw()
Use
<<static>>
Constants
+ MEDIA_FOLDER_LOCATION: string <<const>>
+ WINDOW_WIDTH: int <<const>>
+ WINDOW_HEIGHT: int <<const>>
+ BOARD_X_POSITION: double <<const>>
+ BOARD_Y_POSITION: double <<const>>
+ NUM_OF_LINES_RATE: uint <<const>>
+ NUM_OF_X_CELLS: uint <<const>>
+ NUM_OF_Y_CELLS: uint <<const>>
+ SIZE_OF_BLOCK: double <<const>>
+ SIDEBOARD_X_POSITION: double <<const>>
+ SIDEBOARD_Y_POSITION: double <<const>>
+ HOME_SCENE: int <<const>>
+ SETTING_SCENE: int <<const>>
+ INSTRUCTION_SCENE: int <<const>>
+ GAME_SCENE: int <<const>>
+ ENDING_SCENE: int <<const>>
+ TETROMINO_TYPE_I: int <<const>>
+ TETROMINO_TYPE_J: int <<const>>
+ TETROMINO_TYPE_L: int <<const>>
+ TETROMINO_TYPE_O: int <<const>>
+ TETROMINO_TYPE_S: int <<const>>
+ TETROMINO_TYPE_T: int <<const>>
+ TETROMINO_TYPE_Z: int <<const>>
<<singleton>>
BlockController
- _instance: BlockController
- _blocksIndex: int[,]
- _blocksDisplay: List<Block>
- _score: int
- BlockController()
- BlockController(int numOfXBlocks, int numOfYBlocks)
+ Instance: BlockController << property >>
+ Score: Int << property >>
+ BlocksIndex: int[,] << readonly, property >>
+ BlocksDisplay: List<Block> << readonly, property >>
+ TerminateTetromino(List<Block> tetromino)
+ Update()
+ Draw()
+ Reset()
Block
- _borderColor: Color
- _color: Color
- _xIndex: int
- _yIndex: int
+ Block(Color blockColor)
+ Block(Color blockColor, int xIndex, int yIndex)
+ Block(Color borderColor, Color blockColor, int xIndex, int yIndex)
- BorderRectangle: Rectangle << read-only, property >>
- InnerRectangle: Rectangle << read-only, property >>
+ X: double << readonly, property>>
+ Y: double << readonly, property>>
+ XIndex: int << readonly, property>>
+ YIndex: int << readonly, property>>
+ CanMoveLeft: bool <<readonly, property>>
+ CanMoveRight: bool <<readonly, property>>
+ CanMoveDown: bool <<readonly, property>>
+ Draw()
+ GetIndexes(int xIndex, int yIndex)
+ MoveDown()
Use

Sequence Diagram

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




Singleton
Inherit from
ObserverbleSubject
Implement the
IObserver interface
_currentScene:
GameScene
BlocksController
(BsC)
LineCheckBlocksController
Update()
Update()
Notify()
OnNotify
Loop
All row in
BsC.BlocksIndex[,]
Loop
All column in
BsC.BlocksIndex
[row,]
BlocksIndex
[row, col] == 1
false
true
BlocksIndex.Remove
(YIndex = row)
BsC.Score++
Loop
block in
BsC.BlocksIndex
block.YIndex < row
true
block.MoveDown()
_score = BsC.Score*
(3/ _timeOfSteps*
(600/_gamePlayTime)
return
_score