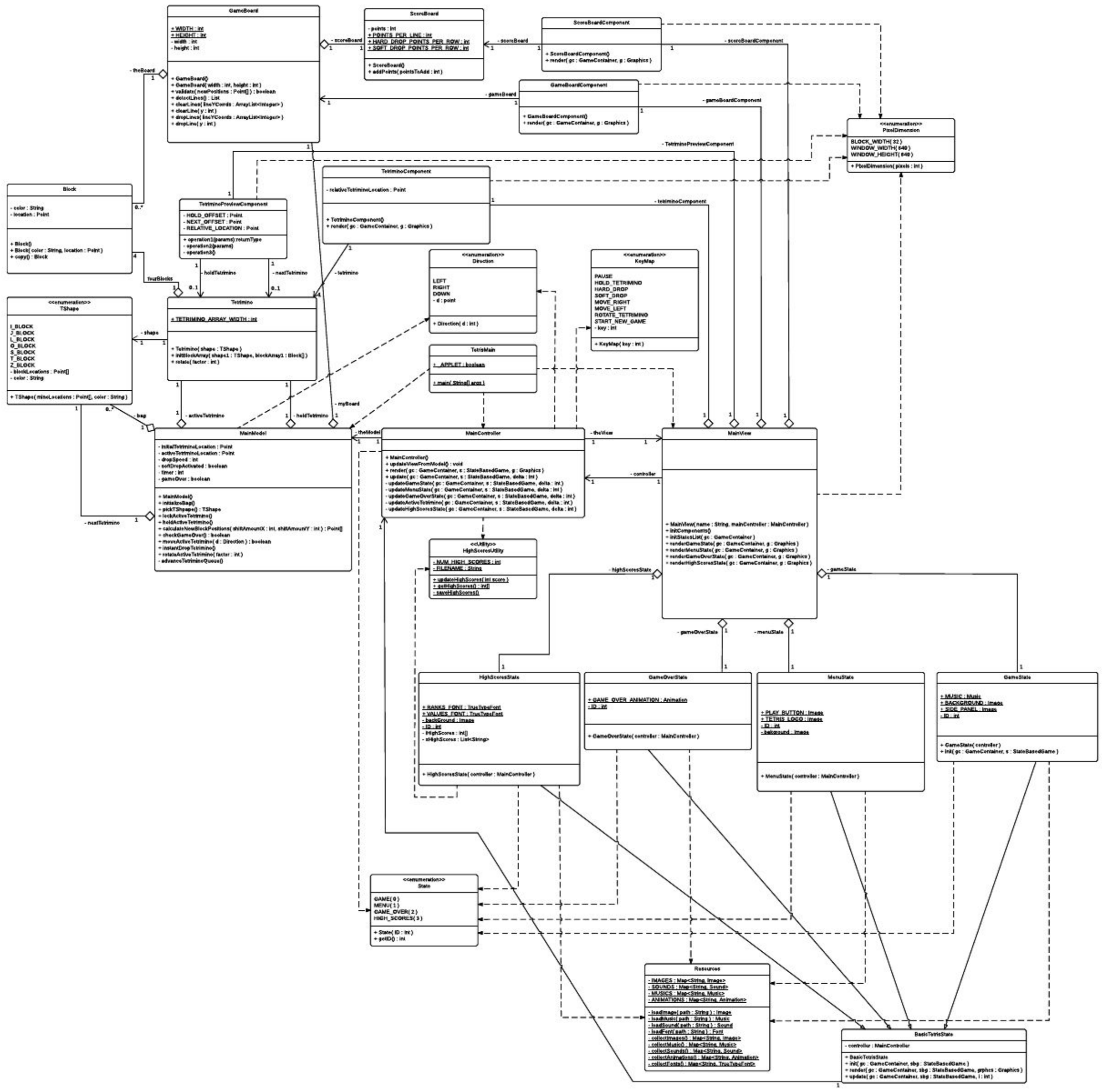
Tetris Design Manual


GameBoard
+ WIDTH : int
+ HEIGHT : int
- width : int
- height : int
+ GameBoard()
+ GameBoard(width : int, height : int)
+ validate(newPositions : Point[]) : boolean
+ detectLines() : List
+ clearLines(lineYCoords : ArrayList<Integer>)
+ clearLine(y : int)
+ dropLines(lineYCoords : ArrayList<Integer>)
+ dropLine(y : int)
ScoreBoard
- points : int
+ POINTS_PER_LINE : int
+ HARD_DROP_POINTS_PER_ROW : int
+ SOFT_DROP_POINTS_PER_ROW : int
+ ScoreBoard()
+ addPoints(pointsToAdd : int)
ScoreBoardComponent
+ ScoreBoardComponent()
+ render(gc : GameContainer, g : Graphics)
GameBoardComponent
+ GameBoardComponent()
+ render(gc : GameContainer, g : Graphics)
<<enumeration>>
PixelDimension
BLOCK_WIDTH(32)
WINDOW_WIDTH(640)
WINDOW_HEIGHT(640)
+ PixelDimension(pixels : int)
Block
- color : String
- location : Point
+ Block()
+ Block(color : String, location : Point)
+ copy() : Block
TetriminoPreviewComponent
- HOLD_OFFSET : Point
- NEXT_OFFSET : Point
- RELATIVE_LOCATION : Point
+ operation1(params):returnType
- operation2(params)
- operation3()
TetriminoComponent
- relativeTetriminoLocation : Point
+ TetriminoComponent()
+ render(gc : GameContainer, g : Graphics)
<<enumeration>>
Direction
LEFT
RIGHT
DOWN
- d : point
+ Direction(d : int)
<<enumeration>>
KeyMap
PAUSE
HOLD_TETRIMINO
HARD_DROP
SOFT_DROP
MOVE_RIGHT
MOVE_LEFT
ROTATE_TETRIMINO
START_NEW_GAME
- key : int
+ KeyMap(key : int)
<<enumeration>>
TShape
I_BLOCK
J_BLOCK
L_BLOCK
O_BLOCK
S_BLOCK
T_BLOCK
Z_BLOCK
- blockLocations : Point[]
- color : String
+ TShape(minoLocations : Point[], color : String)
Tetrimino
+ TETRIMINO_ARRAY_WIDTH : int
+ Tetrimino(shape : TShape)
+ initBlockArray(shape1 : TShape, blockArray1 : Block[])
+ rotate(factor : int)
TetrisMain
+ _APPLET : boolean
+ main(String[] args)
MainModel
- initialTetriminoLocation : Point
- activeTetriminoLocation : Point
- dropSpeed : int
- softDropActivated : boolean
- timer : int
- gameOver : boolean
+ MainModel()
+ initializeBag()
+ pickTShpape() : TShape
+ lockActiveTetrimino()
+ holdActiveTetrimino()
+ calculateNewBlockPositions(shiftAmountX : int, shiftAmountY : int) : Point[]
+ checkGameOver() : boolean
+ moveActiveTetrimino(d : Direction) : boolean
+ instantDropTetrimino()
+ rotateActiveTetrimino(factor : int)
+ advanceTetriminoQueue()
MainController
+ MainController()
+ updateViewFromModel() : void
+ render(gc : GameContainer, s : StateBasedGame, g : Graphics)
+ update(gc : GameContainer, s : StateBasedGame, delta : int)
- updateGameState(gc : GameContainer, s : StateBasedGame, delta : int)
- updateMenuState(gc : GameContainer, s : StateBasedGame, delta : int)
- updateGameOverState(gc : GameContainer, s : StateBasedGame, delta : int)
- updateActiveTetrimino(gc : GameContainer, s : StateBasedGame, delta : int)
- updateHighScoresState(gc : GameContainer, s : StateBasedGame, delta : int)
MainView
+ MainView(name : String, mainController : MainController)
+ initComponents()
+ initStatesList(gc : GameContainer)
+ renderGameState(gc : GameContainer, g : Graphics)
+ renderMenuState(gc : GameContainer, g : Graphics)
+ renderGameOverState(gc : GameContainer, g : Graphics)
+ renderHighScoresState(gc : GameContainer, g : Graphics)
<<Utility>>
HighScoresUtility
- NUM_HIGH_SCORES : int
- FILENAME : String
+ updateHighScores(int score)
+ getHighScores() : int[]
- saveHighScores()
HighScoresState
+ RANKS_FONT : TrueTypeFont
+ VALUES_FONT : TrueTypeFont
- backGround : Image
- ID : int
- iHighScores : int[]
- strHighScores : List<String>
+ HighScoresState(controller : MainController)
GameOverState
+ GAME_OVER_ANIMATION : Animation
- ID : int
+ GameOverState(controller : MainController)
MenuState
+ PLAY_BUTTON : Image
+ TETRIS_LOGO : Image
- ID : int
- bakground : Image
+ MenuState(controller : MainController)
GameState
+ MUSIC : Music
+ BACKGROUND : Image
+ SIDE_PANEL : Image
- ID : int
+ GameState(controller)
+ init(gc : GameContainer, s : StateBasedGame)
<<enumeration>>
State
GAME(0)
MENU(1)
GAME_OVER(2)
HIGH_SCORES(3)
+ State(ID : int)
+ getID() : int
Resources
- IMAGES : Map<String, Image>
- SOUNDS : Map<String, Sound>
- MUSICS : Map<String, Music>
- ANIMATIONS : Map<String, Animation>
- loadImage(path : String) : Image
- loadMusic(path : String) : Music
- loadSound(path : String) : Sound
- loadFont(path : String) : Font
- collectImages() : Map<String, Image>
- collectMusic() : Map<String, Music>
- collectSounds() : Map<String, Sound>
- collectAnimations() : Map<String, Animation>
- collectFonts() : Map<String, TrueTypeFont>
BasicTetrisState
- controller : MainController
+ BasicTetrisState
+ init(gc : GameContainer, sbg : StateBasedGame)
+ render(gc : GameContainer, sbg : StateBasedGame, grphcs : Graphics)
+ update(gc : GameContainer, sbg : StateBasedGame, i : int)
- scoreBoard
- theBoard
- gameBoard
- scoreBoardComponent
- gameBoardComponent
- TetriminoPreviewComponent
- tetriminoComponent
fourBlocks
- holdTetrimino
- nextTetrimino
- tetrimino
- shape
- bag
- activeTetrimino
- holdTetrimino
- myBoard
- nextTetrimino
- theModel
- theView
- controller
- highScoresState
- gameState
- gameOverState
- menuState

Introduction:

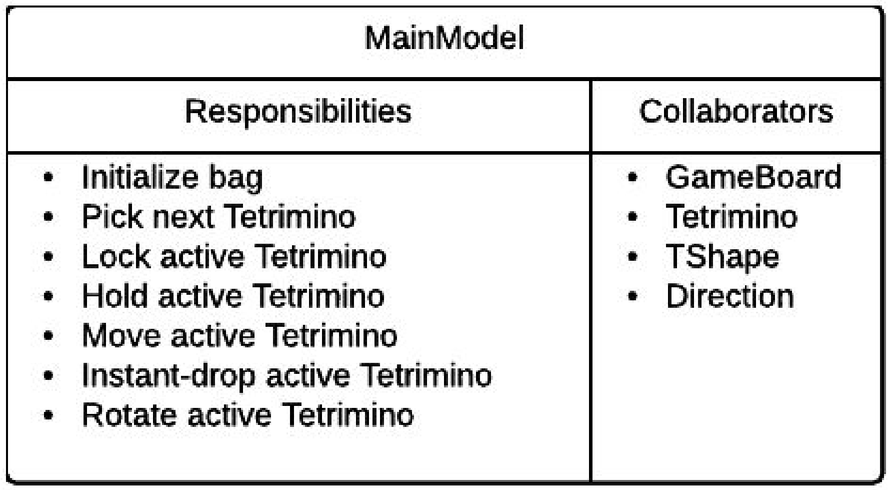
This Tetris clone is structured according to the MVC design pattern. As Slick2D prevents a perfect implementation of MVC, delegation was used to bridge the gaps. Namely, each of the states (which extend Slick's BasicGameState class) hold a reference to an existing MainController. The subsequent update() and render() methods are passed off to the controller's corresponding methods. When rendering a state, the controller passes this onto its MainView object. With some adjustment to accommodate the hybrid view/controller nature of the game state, we were able to adhere to MVC to the best of our ability. Namely, the entire internal state of the game (i.e. the player's score, number of lines cleared, active Tetrimino, held Tetrimino, and so on) is firmly encapsulated by the MainModel, all render() methods are owned by MainView and its component instance variables, and the logic to synchronize the two are executed by MainController.

TetrisMain:

There are 3 main classes: MainModel, MainController, and MainView, each located in the tetris.model, tetris.controller, and tetris.view packages, respectively. As expected of the MVC design pattern, the MainModel class is instantiated and represents the state of the game's data and related functions. The MainController class executes function calls on MainModel, causing the game to operate and call functions in the MainView class to update the graphical components presented to the user. TetrisMain is the true "main" class which simply instantiates a model, view and controller, along with an AppGameContainer (passed to the render() and update() methods as "gc" for game container). The dimensions of the window are set for the game container; finally, the game is started.

MainModel

The MainModel class stores data relevant to the game. It instantiates a GameBoard, a Tetrimino for each the activeTetrimino and the nextTetrimino, and a TShape for the nextTetrimino. The MainModel also keeps track of information regarding the active Tetrimino's location, the collection of shapes from which the next Tetrimino will be selected, whether or not the Tetrimino can be held again, drop speed, whether a soft drop is toggled, the game timer, the number of lines cleared in the game, and whether or not the game has been lost.



MainModel	
Responsibilities	**Collaborators**
• Initialize bag • Pick next Tetrimino • Lock active Tetrimino • Hold active Tetrimino • Move active Tetrimino • Instant-drop active Tetrimino • Rotate active Tetrimino	• GameBoard • Tetrimino • TShape • Direction

MainModel also contains methods that manipulate the game data, as alluded to in the diagram above.

GameBoard

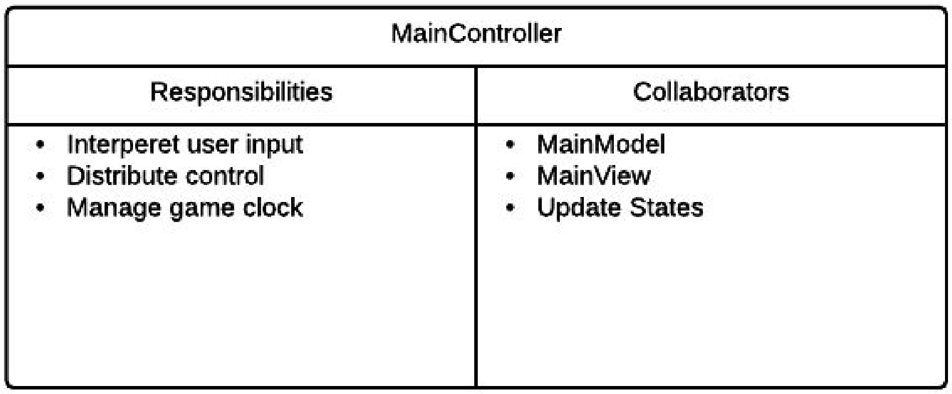
The GameBoard represents the game's "Tetris Board". This is where all the real gameplay occurs. The GameBoard consists of an array of Block objects and contains functions that manipulate the array. The GameBoard also instantiates a ScoreBoard.

Tetrimino

Tetriminos contain a 2 dimensional array of Block objects that can be "rotated" with the Tetrimino's rotate function. The Tetrimino is constructed using a TShape enumerated constant which defines the shape of the Tetrimino. The Direction enum is used to define the possible directions in which the Tetrimino can be moved.

MainController

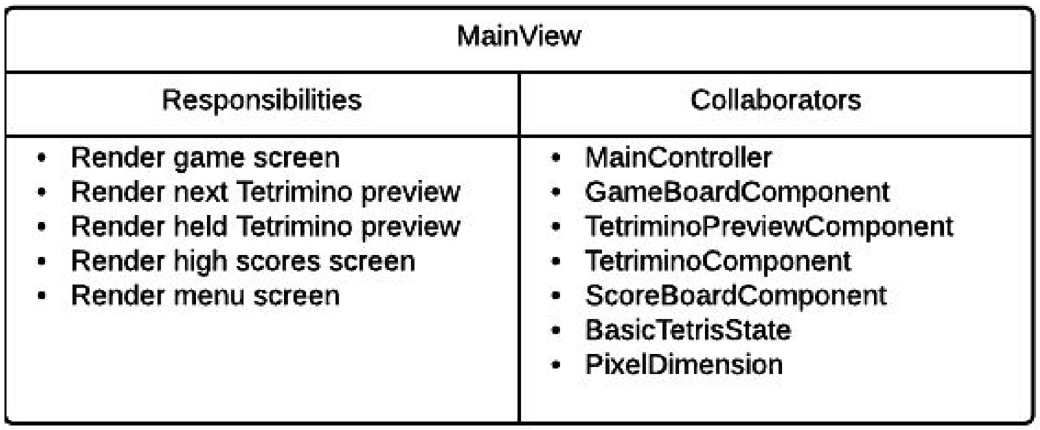
The MainController class uses the functions of MainModel to manipulate the game and the functions of MainView to render the the individual game states via delegation. In MainController, inputs are bound to actions and are dependent on the current game state, detailed in Game States below. The Keymap is used to define the key bindings of the game; Each user action is associated with a particular input.



MainController	
Responsibilities	**Collaborators**
• Interperet user input • Distribute control • Manage game clock	• MainModel • MainView • Update States

MainView

The MainView instantiates each of the component classes and draws them.



MainView	
Responsibilities	**Collaborators**
• Render game screen • Render next Tetrimino preview • Render held Tetrimino preview • Render high scores screen • Render menu screen	• MainController • GameBoardComponent • TetriminoPreviewComponent • TetriminoComponent • ScoreBoardComponent • BasicTetrisState • PixelDimension

Game States

A game state can be thought of as a particular “screen” present in the game. For example, the menu screen and game screen are game states. Slick2D relies on states (which are defined in the tetris.view.GameStates package as separate state classes) to determine what to render and how inputs are defined and whether or not they are active. For example, it definitely isn't ideal to interpret the user hitting "space" while in the HighScoresState to instantly drop the Tetrimino loaded into the inactive GameState.

The functions update*GAME_STATE*(GameContainer gc, StateBasedGame s, int delta), where *GAME_STATE* is the name of some game state, are called to check for input and execute the appropriate commands in order to update the game while it’s in that particular state.