CPSC 224 Final Project

Project Plan

March 3, 2024

Java Tetris

Team Tetris

Prepared by: Nick DeYoung, Mingze Zhang, Jack Ou

1 Project Overview

1.1 Project Summary

In Tetris, players are tasked with manipulating a series of falling geometric shapes, called tetrominoes, as they descend from the top of the playing field. The objective is to strategically rotate and position these tetrominoes to create complete horizontal lines without any gaps. When a line is formed, it disappears, creating space for more pieces to fall. As the game progresses, the tetrominoes fall at an increasing speed, challenging players to think quickly and make split-second decisions.

There is no definitive condition for winning a game of Tetris. Tetris is a game where the player loses by letting the tetrominoes stack to the top of the playing grid. The main motivation to keep the game of Tetris alive is to keep clearing lines and beating a previous top score set by another player. The unstated condition of winning a game of Tetris is to beat another player's score.

2 Project Requirements

2.1 Major Features

Feature	Description
Moving the block horizontally	-High Priority. -The player will be able to move the block along the x axis by pressing either the 'a' and 'd' keys or the 'leftArrow' and 'rightArrow' keys. -Occurs within the bounds of the playing field, requires key input, if the key is held it should continue moving horizontally at a set speed. If against a wall, block does not move. -The active player is involved, so is the active piece, the grid array in the background, the borders of the wall, the timer. -No Output
	-High Priority. -The player will be able to hold the down the 's' key or the 'downArrow' key as long as they want until they reach the bottom or another solid piece. For each space they move down in this way, the score is increased.

Moving the block downward	-Occurs until the block reaches the bottom of the screen, the score increases by a set amount during this time. -The active player, the active block, the grid background, the score, and the highest piece that contacts the active piece are involved. -The output is an increase in the score based on how long the button is held.
Rotate the block	-High Priority. -The player will be able to rotate the block using the 'O' and 'P' Keys. The O key rotates counter-clockwise and the P key rotates clockwise. -When a valid key is pressed, the block rotates in the desired direction based on a rotation algorithm that reassigns block locations. It needs to stop itself from happening if a piece would have been relocated outside of the playing field or was already occupied by a solid piece at the bottom. -No output
Splash Screen	-High Priority -The Game will load up a screen before the game itself loads giving information like a well-designed Tetris logo, who made the project and when. -The Splash screen takes no inputs and has no needs, it is opened as soon as the program is run and does not come up again -No player or object interacts with the splash screen except for a timer which keeps the screen open until it is closed. -There could be a state of whether the screen is open or not, otherwise the splash screen affects nothing.
Win/Lose Screen	-High Priority -The game will display a screen when the player has succumbed to the tetrominoes that displays their score (or number of pieces used).
Game Board Screen	-High Priority - The function shall present the game screen interface with all necessary game information, allowing players to interact and play Tetris. It provides the main gameplay experience, including moving pieces and clearing rows. - Inputs include game rules, player preferences (e.g., difficulty level), and player actions (e.g., moving the pieces). Valid ranges for rotation outcomes and score calculations must be defined. Special cases involve handling game interruptions (e.g., pausing) and ensuring accessibility for players with various abilities. - The GameApp object manages the game screen and its components, blocks, grid, and player controls. Players actively engage with the game screen, making decisions and observing outcomes. - This requirement outputs the interactive game screen interface, updating game state based on player actions and providing feedback on scores and game progression.
Setting Information	-Medium Priority -The function shall provide a settings screen accessible from the game interface, allowing players to adjust game parameters such as sound preferences, game difficulty,

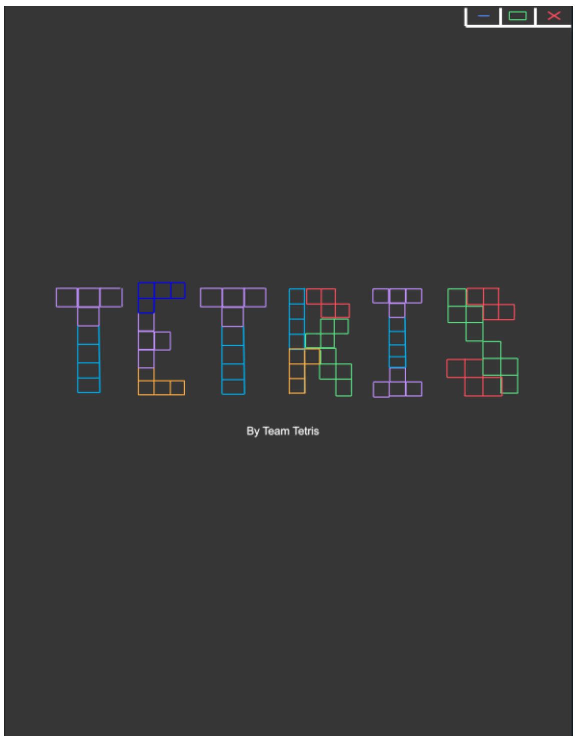
	and player names. It offers customization options to tailor the gameplay experience to individual preferences. Inputs include player preferences for sound settings (e.g., volume, mute), game difficulty levels (e.g., easy, medium, hard), and player name customization. Valid ranges for volume levels and difficulty settings should be defined. Special cases involve handling compatibility with different audio output devices and ensuring user-friendly input mechanisms for name customization. The GameApp object integrates the settings screen into the game interface, allowing players to access and modify game parameters. Players interact directly with the settings screen, adjusting preferences according to their preferences. This requirement outputs the settings screen interface, enabling players to modify game parameters. Upon confirmation of settings adjustments, the game state is updated accordingly, reflecting changes in gameplay experience such as difficulty adjustments or customized player names.
Scoring System	-Medium Priority The function shall implement a comprehensive scoring system that accurately evaluates player actions and assigns scores based on Tetris game rules. Inputs include line clearing, and adherence to Tetris scoring rules. Special cases involve handling complex scoring scenarios, clearing multiple lines and increasing block speed. The ScoreManager object within the GameApp oversees the scoring system, tracking player scores and updating game state accordingly. Players interact indirectly with the scoring system through their gameplay decisions and observations of score updates. Additionally, the scoring system may trigger visual feedback to players, highlighting successful scoring clears.
Level Progression	-Low Priority The function shall incorporate a level progression system that dynamically adjusts game difficulty as players advance, providing an increasingly challenging experience. Inputs include player performance metrics such as score thresholds or completed objectives, which trigger level advancements. Special cases involve balancing difficulty progression to ensure a gradual increase in challenge without overwhelming players. The LevelManager object within the GameApp oversees the level progression system, monitoring player performance and adjusting game parameters accordingly. This requirement sets the current game level within the game state, influencing gameplay elements such as speed or complexity of gameplay mechanics. Additionally, it may provide visual or auditory cues to inform players of their progression and impending challenges as they advance through levels.

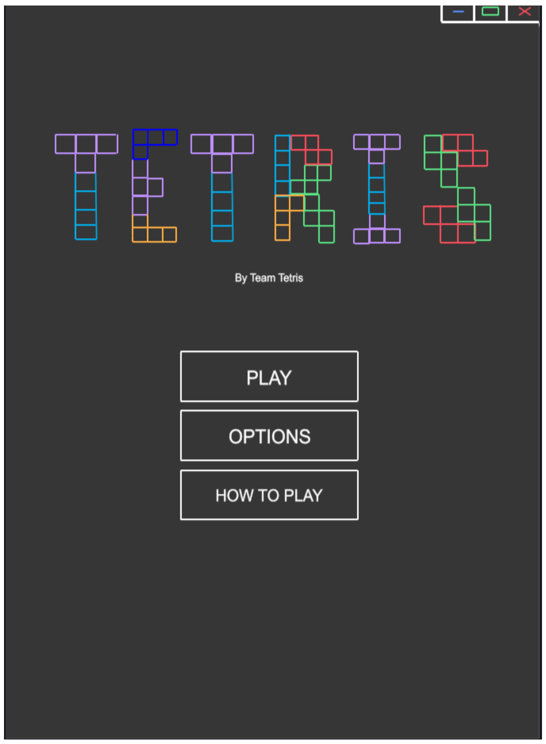
Clearing Lines	-Medium Priority The function shall implement a line clearing mechanism that removes completed lines of blocks from the game board, rewarding players with points and creating space for further gameplay. It maintains the flow of gameplay by incentivizing efficient block placement and strategic thinking. Inputs include completed lines of blocks identified by their position on the game board. Valid ranges for the number of lines cleared per action should be defined. Special cases involve handling simultaneous line clears and cascading effects from cleared lines. The BoardManager object within the GameApp coordinates the line clearing process, updating the game board and scoring system accordingly. Players interact directly by placing blocks strategically to create line-clearing opportunities. This requirement outputs the removal of completed lines from the game board, adjusting the game state to reflect the updated board configuration. It sets the score based on the number of lines cleared, rewarding players for their efficiency and strategic planning.
Music System	-Low Priority The function shall integrate music into the game environment, enhancing immersion and player experience. Inputs include a selection of background music tracks suitable for the game's theme and mood. Valid ranges for volume levels should be defined. The AudioManager object within the GameApp manages the playback of music tracks, adjusting volume and track selection as necessary. This requirement outputs the continuous playback of selected music tracks during gameplay.
Pausing the game	-Low Priority The function shall enable players to pause the game at any point during gameplay, allowing them to temporarily suspend gameplay activity and attend to other matters. It offers flexibility and convenience, ensuring players can resume gameplay from where they left off without losing progress. Inputs include player input to activate the pause feature, typically through a designated button or menu option. Valid ranges for pause durations should be defined. Special cases involve handling interruptions gracefully, such as pausing during critical gameplay moments or while displaying important information. The GameManager object within the GameApp manages the pausing functionality, halting game processes and displaying the pause menu. Players directly interact with the pause feature to control the flow of gameplay. This requirement outputs the pause menu interface, providing options for players to resume gameplay, adjust settings, or exit the game. It sets the game state to a

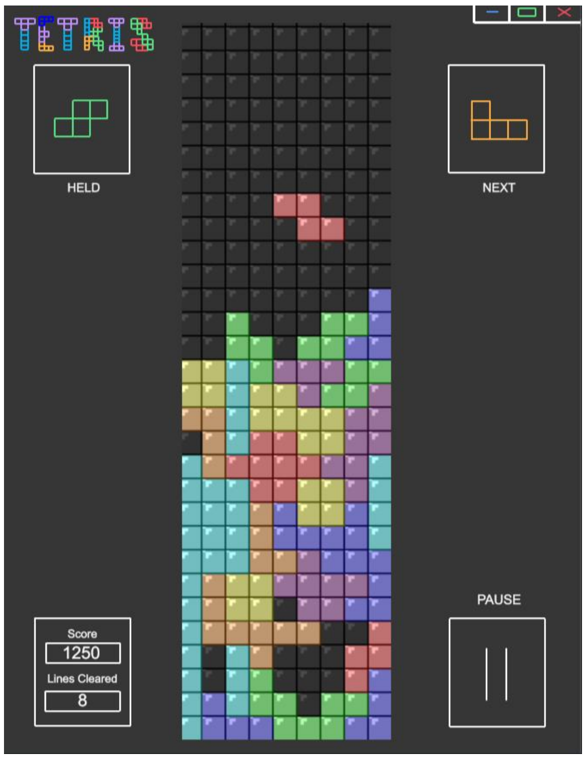
	paused state, suspending all active gameplay processes until the player chooses to resume.
Holding/displaying blocks	-Medium Priority The function shall allow players to hold a block temporarily, storing it for later use, and display the next upcoming block to inform strategic planning. It enhances player decision-making by providing insight into future gameplay possibilities. Inputs include player input to activate the block holding feature and the upcoming block to be displayed. Valid ranges for held block display size should be defined. Special cases involve managing conflicts between held blocks and current gameplay mechanics, such as restrictions on holding blocks during certain gameplay phases. The BlockManager object within the GameApp oversees the holding and displaying of blocks, managing storage and visual representation. Players interact directly with the block holding feature, making strategic decisions about when to hold or release blocks. This requirement outputs the visual display of the next upcoming block and the ability to hold a block, storing it for future use. It sets the game state to reflect the held block status, updating the display to indicate whether a block is currently being held or released for gameplay

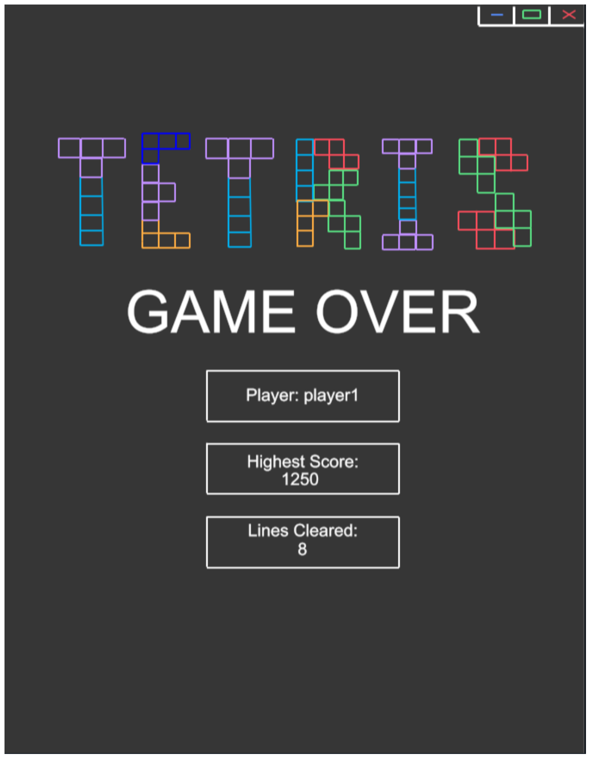
3 Project Game Design

3.1 Initial User Interface Design

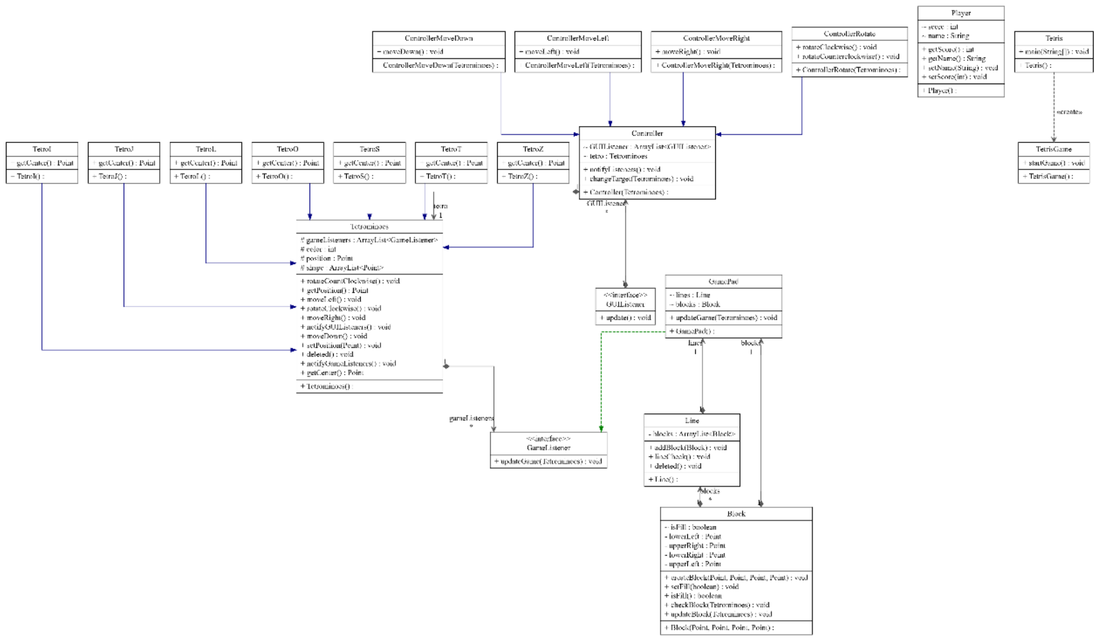


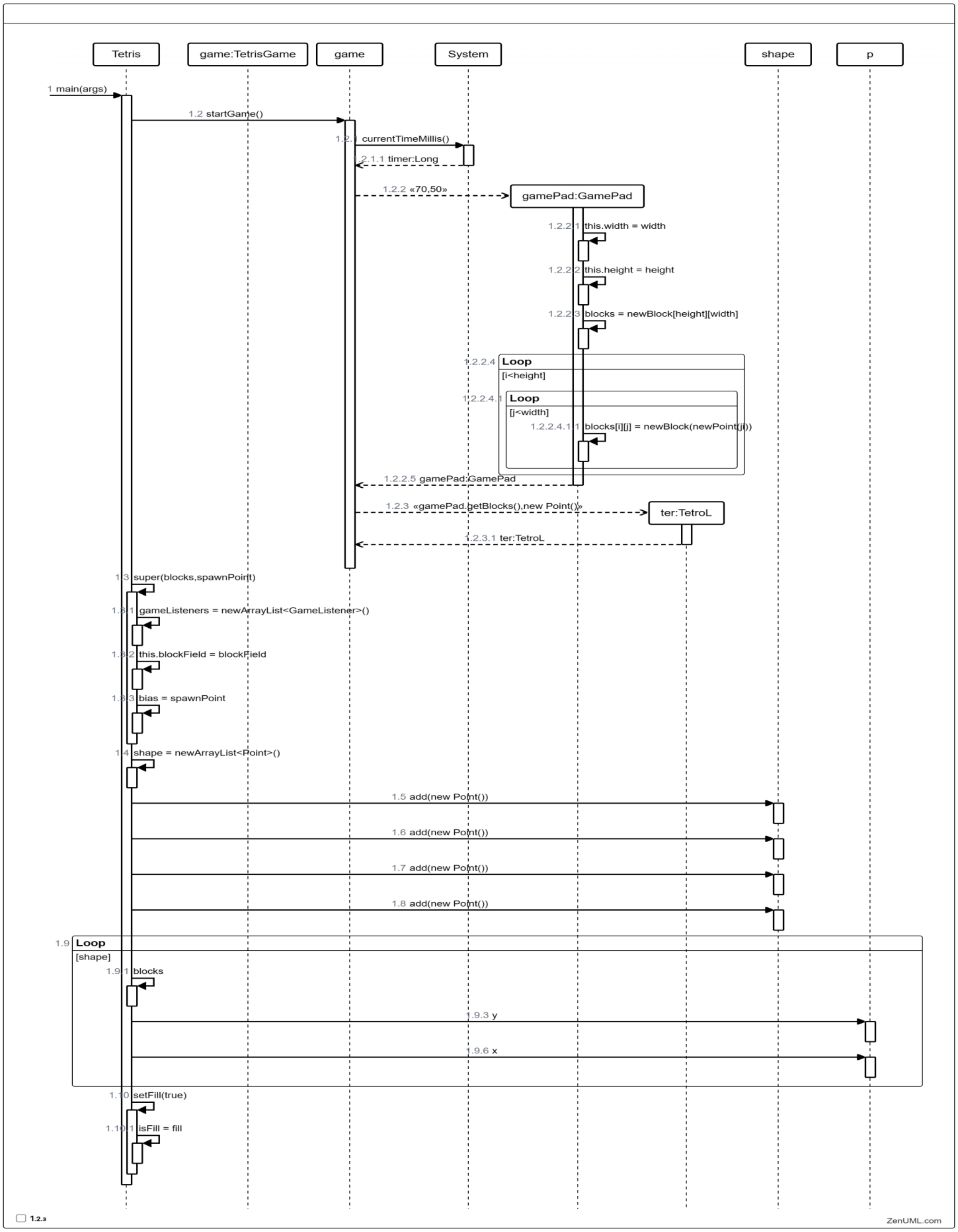






3.2 Initial Software Architecture


ControllerMoveDown
− moveDown() : void
ControllerMoveDown(Tetrominoes) :
ControllerMoveLeft
− moveLeft() : void
+ ControllerMoveLeft(Tetrominoes) :
ControllerMoveRight
+ moveRight() : void
+ ControllerMoveRight(Tetrominoes) :
ControllerRotate
+ rotateClockwise() : void
+ rotateCounterclockwise() : void
+ ControllerRotate(Tetrominoes) :
Player
− score : int
− name : String
+ getScore() : int
+ getName() : String
+ setName(String) : void
+ setScore(int) : void
+ Player() :
Tetris
+ main(String[]) : void
+ Tetris() :
«creates»
TetrisGame
+ startGame() : void
+ TetrisGame() :
Controller
− GUIListener : ArrayList<GUIListener>
− tetro : Tetrominoes
+ notifyListeners() : void
+ changeTarget(Tetrominoes) : void
+ Controller(Tetrominoes) :
GUIListener
*
TetroI
+ getCenter() : Point
− TetroI() :
TetroJ
+ getCenter() : Point
+ TetroJ() :
TetroL
+ getCenter() : Point
+ TetroL() :
TetroO
+ getCenter() : Point
+ TetroO() :
TetroS
+ getCenter() : Point
+ TetroS() :
TetroT
+ getCenter() : Point
− TetroT() :
TetroZ
+ getCenter() : Point
− TetroZ() :
tetro
1
Tetrominoes
# gameListeners : ArrayList<GameListener>
# color : int
# position : Point
# shape : ArrayList<Point>
+ rotateCountClockwise() : void
+ getPosition() : Point
+ moveLeft() : void
+ rotateClockwise() : void
+ moveRight() : void
+ notifyGUIListeners() : void
+ moveDown() : void
+ setPosition(Point) : void
+ deleted() : void
+ notifyGameListeners() : void
+ getCenter() : Point
+ Tetrominoes() :
gameListeners
*
<<interface>>
GameListener
+ updateGame(Tetrominoes) : void
<<interface>>
GUIListener
+ update() : void
GamePad
− lines : Line
− blocks : Block
+ updateGame(Tetrominoes) : void
+ GamePad() :
lines
1
blocks
1
Line
- blocks : ArrayList<Block>
+ addBlock(Block) : void
+ lineCheck() : void
+ deleted() : void
+ Line() :
blocks
*
Block
− isFill : boolean
- lowerLeft : Point
- upperRight : Point
- lowerRight : Point
- upperLeft : Point
+ createBlock(Point, Point, Point, Point) : void
+ setFill(boolean) : void
+ isFill() : boolean
+ checkBlock(Tetrominoes) : void
+ updateBlock(Tetrominoes) : void
+ Block(Point, Point, Point, Point) :


Tetris
game:TetrisGame
game
System
shape
p
1 main(args)
1.2 startGame()
1.2.1 currentTimeMillis()
1.2.1.1 timer:Long
1.2.2 «70,50»
gamePad:GamePad
1.2.2.1 this.width = width
1.2.2.2 this.height = height
1.2.2.3 blocks = newBlock[height][width]
1.2.2.4 Loop
[i<height]
1.2.2.4.1 Loop
[j<width]
1.2.2.4.1.1 blocks[i][j] = newBlock(newPoint(ji))
1.2.2.5 gamePad:GamePad
1.2.3 «gamePad.getBlocks(),new Point()»
ter:TetroL
1.2.3.1 ter:TetroL
1.3 super(blocks,spawnPoint)
1.3.1 gameListeners = newArrayList<GameListener>()
1.3.2 this.blockField = blockField
1.3.3 bias = spawnPoint
1.4 shape = newArrayList<Point>()
1.5 add(new Point())
1.6 add(new Point())
1.7 add(new Point())
1.8 add(new Point())
1.9 Loop
[shape]
1.9.1 blocks
1.9.3 y
1.9.6 x
1.10 setFill(true)
1.10.1 isFill = fill
1.2.3
ZenUML.com

4 Project Schedule

Milestone	Description	Target Completion Date
Gameboard functioning	Have a visualized game board which creates a grid of pleasant looking blocks, creates and visualizes an active block (controlled by the player). The active block can move left and right, rotate, and stop when it reaches the bottom of the screen or another block already at the bottom. For this we will need to research: -Methods of representing a "block" in our grid. -How to "rotate" a block as its falling and ensure two blocks do not end up in the same location -How to display pleasant images on top of the grid to improve player experience	April 1st.
Project Plan Due	The project plan document is complete and turned in including all functional requirements for the project, a detailed UML diagram and clean, well done drawings of all UI panels. All team members contributed significantly and the document was reviewed before submition	April 3rd
Multiple UI menus functioning	Have 4 distinct user interface menus which are able to display the information required for a person to play the game and also to capture all of our functional requirements in an intuitive layout. Includes updates to the game board such as ghost	April 10th

	blocks to display where the piece will end up. For this we will need to research: -Switching visual Gui screens and the commands to do so -Nicely displaying the information we have using Jframe and layeredJpanels -How to pause the game timer when the screen is switched or the pause button is hit, and how to keep the positions of the game board the same	
All UI aspects working correctly	Every piece of the UI is complete and displayed in the correct location according to our drawings. The game can pause and resume correctly. The game board is neat and clean and displays elements in a pleasing way. There are still possibly some minor bugs to be ironed out in the final week. For this we will need to research: -Tetris scoring and how to code it algorithmically -Pleasing UI design and how to make a program run smoothly with nice visual elements -How to play music and adjust volume from within a program.	April 17th
Code Complete	The code is complete, functions without issues and performs all the functional requirements laid out in our functional requirements table.	April 25th
Team Presentations	The code is ready to be presented, clean looking and readable accompanied by a powerpoint/other presentation method to display to the class and others who may not be able to code exactly what we did to make our program function correctly.	April 29th

Final Report Due	A final report is typed up which fills out all requirements, has been edited by all members of the team and looked over for accuracy and completion before its submition	May 9th