BILKENT UNIVERSITY

CS 319

Object Oriented Software Engineering

ANALYSIS REPORT ITERATION 2

16 April 2019

QUADRILLION

by The Group

Ana Peçini

Krisela Skënderi

Muhammad Hammad Malik

Ünsal Öztürk

Table of Contents

1. Introduction

Quadrillion is a one-player board game designed by SmartGames, a company that develops IQ puzzles with innovative game mechanics [1] that are designed to stimulate cognitive skills such as concentration, problem solving and logic [2]. What makes this board game stands out from other ones of the same category is that even with the constraint of using the same grid parts and puzzle pieces over and over again, there exist as many as 638 to 976 different start positions. [3]

The board of the game consists of 4 black and white magnetic grid pieces that can be rotated or flipped and then arranged together in any combination. The game includes 12 different puzzle pieces [2]. The aim of Quadrillion is to arrange the 12 different puzzle pieces in such a way that they fit perfectly in the grid without leaving any empty spaces and without overlapping any of the blocked positions on the grid, which are fixed positions indicated by the opposite colour to that of the grid. The original game comes with a booklet of grid arrangements that are sorted according to their difficulty level. Each arrangement suggested in the booklet can have at most one solution, but the player can come up with his own grid designs that, if following a few simple guidelines and specifications, are guaranteed to have at least one solution but may have up to thousands [3].

The Group's digitalized Quadrillion, called Gazillion, is a roguelike version that enhances the dynamics of the original game by adding new immersive features, while preserving the core mechanics of the board game. The features include:

- multiple modes with different goals and interactive maps
- player inventory, health and budget
- time constraints for different levels
- shop entity where different themes and power ups can be purchased, etc.

2. Overview

2.1 General Information

We propose a system that digitalizes Quadrillion, called Gazillion. The system will simulate the core aspects of Quadrillion gameplay, as well as extended functionality based on top of the game. The game will be a desktop based Java application. There will be three game modes that revolve around providing a different experience of Quadrillion to the player. These modes are:

- Treasure Mode:

This mode is a “game within a game”. The user will be able to see a map with nodes, in which there are randomly generated games of Quadrillion. The levels are arranged in a grid, and the user can click on an unlocked level to play a round of Quadrillion. Once the level is completed within the time constraint, the levels that are connected to the completed level are unlocked, and the user may play those levels as well. The purpose of this game mode is to collect all Quadrillion pieces that are hidden in certain levels on the map, and to solve a final puzzle of Quadrillion once all the pieces have been gathered.

- Ladder Mode:

The Player will be provided randomly generated and unlabeled Quadrillion games (in terms of difficulty) one after another. The health mechanic is absent from this game mode. The Player may not use any power ups in this mode. The Player will be awarded 10 gold coins after each completed level.

- Level Mode:

In the Levels mode, the player can choose among pre-made and “difficulty labeled” levels and play them without a time constraint. The levels are initially locked except the first one, and the player should play levels one by one to unlock harder levels. An initial completion of a level provides the player coins. The “time” and “health” mechanics are absent in this game mode. The purpose of this mode is to simply complete the round of Quadrillion, without any time or failure concerns.

2.2 Game Mode Specifications

2.2.1 Treasure Mode

- Initially only the levels in the 4 corners of the map are available for the Player to play.
- The rest of the levels are locked and the player unlocks new levels by playing a level which is adjacent to the completed level.
- Initially, the player will have five health points.
- The player will have a time constraint to solve individual levels and he will lose health points if he is unable to.
- The player can gather power-ups by successfully completing levels or purchasing them in the shop.

- The treasure mode allows the player to use three types of power-ups (health, time, treasure hints).
- The time power-up gives player extra time (1 minute) during a Quadrillion game.
- The health power-up restores one health point to the player.
- The hint power up gives the player a hint regarding the locations of Quadrillion pieces in the map.
- The player may use the hint power-up in the map screen, and the time power-up and health may be used during the game of Quadrillion.
- If the user runs out of health at any point the Quadrillion game is over and the Player loses. He has to restart the game from the beginning.
- Once the player has collected all the pieces, the player plays a final round of Quadrillion, which has a time constraint. If the player is able to win the final round of Quadrillion, the player is considered to have won the treasure mode.

2.2.2 Ladder Mode

- The player is greeted by an "intermission" message upon entering the ladder mode, displaying the amount of time the player has for the next round.
- If the player is able to solve the Quadrillion level before the timer runs out, the player is again displayed the "intermission" message for fifteen seconds and the number of rounds the player has won thus far. This goes on until the user runs out of time.
- When the player runs out of time, the game of Quadrillion is pre-emptively terminated and the player is displayed for a final time the number of levels he has completed. The player is awarded coins based on the number of levels completed.
- In this mode the player may only use the time power-up, as the health and treasure hint mechanics are disabled. He may use as many time power-ups as desired.

2.2.3 Level Mode

- In this mode, the player is able to choose from an array of pre-made levels.
- Initially, all the levels are locked but the first one.
- The player plays the first level, and unlocks the next one.
- The player is awarded five coins for the completion of every level.
- The levels do not have any form of time or health constraints.

3. Functional Requirements

The game will provide the following functionalities to the Player:

3.1 Playing a round of Quadrillion:

Quadrillion comes in 3 different modes, all of which include the basic gameplay of the traditional board game. The original game grid is composed of 4 4x4 grids arranged in a particular shape with fixed blocker positions. The 12 quadrillion pieces are provided to the Player. The common functionalities with our system are:

- Place Quadrillion pieces on the game board
- Rotate Quadrillion pieces by 90 degrees, clockwise and counter-clockwise.
- Flip Quadrillion pieces (180 degrees)
- View the remaining time for the round
- View the remaining health, if included in the mode
- Use power-ups according to the mode
- Give up on the round / leave the game

Additionally, Player has a power up inventory, a health bar and coins. Initially Player's inventory is initialized with 5 power ups and 5 health units. Depending on the selected mode, a timer will keep track of the elapsed time for the Quadrillion puzzle from the moment when the Player enters the screen until the puzzle is completed. If the Player is not able to finish the round in time he may be punished by losing health points depending on the mode. The Player may use a "time" power up at any point during the round to increase remaining time. When the Player completes a level, he is awarded with virtual currency (coins) which serve to buy power ups, themes, and hints.

3.2 Game modes:

The system will provide the Player with a choice between 3 game modes in the Main Menu as explained above:

- Levels mode
- Ladder mode
- Treasure mode

3.3 Settings:

The system will provide the Player the ability to adjust certain settings related to the game. The Player may access this menu by clicking on the "Settings" button in the Main Menu. The provided functionalities are:

- Change the volume level
- Change the theme
- Mute the sound

3.4 Shop:

The system will provide the Player the possibility of buying power-ups and themes in exchange for coins. The Player may access the shop by clicking on the "Shop" button in the Main Menu. The shop provides the following functionalities:

- Buy "health" power up
- Buy "time" power up
- Buy "hint" power up
- Buy theme

3.5 How to Play:

The system will allow the Player to view the rules of the game and instructions in the form of an image. Player has to click on the "Instructions" button in the Main Menu.

3.6 Credits:

The Player will be able to see the developers, and the people who were involved in the development of the game by clicking on the "credits" button.

4. Non-functional Requirements

4.1 Accessibility:

The game should be easy to distribute and download. Given a certain hardware specification it should not take more than 3 minutes to install the game.

4.2 Usability:

- The average Player should be acquainted with the basics of the game and the system in a time interval of no more than 30 minutes with or without reading the game manual. The game should also provide the Player full control over the Quadrillion pieces: the Player should be easily able to rotate, flip, and place pieces, and the these operations should be handled by both mouse and keyboard, so that the Player can perform these actions in under a second on average.
- The average time constraint for a level is 10 minutes, so there is enough time to complete the levels, but not too much, in order to keep the game challenging.
- The game should provide the Player enough resources (i.e.) 5 coins per level to reduce the challenge posed by the difficulty of the levels.

4.3 Portability:

The system should be easily portable to other platforms, such as OSX and Linux. This will be achieved by developing the system on a platform which runs on all of these platforms.

4.4 Performance:

The system should be responsive to the Player's actions and should run smoothly to maintain the Player experience above a certain standard. These standards are:

- Smooth motion and animations: the frame rate should be at least 40 frames per second, or should be at least 24 frames per second, given a motion blur implementation.
- The system should maintain responsiveness by reducing latency between its logical calculations and raster/graphical operations to less than 40 milliseconds. This is the case since some of the modes in the games have time constraints.
- The game should not load, launch, or take any action that will make the Player wait (e.g. generating random levels of Quadrillion) for more than a fixed amount of time.

The game modes should take no more than 5 seconds to load after Player has made his choice.

5. Pseudo Requirements

- The system will be fully developed using Java.
- The system will be compiled into an independent .jar executable.
- The system will run on the Java Virtual Machine.
- The system will not require internet connectivity.

6. System Requirements

- The game must take up no more than 512 MB disk space.
- The game must run on any machine that supports JVM.
- The game must not consume 80% of the resources(memory, CPU clock cycles) of a commercially available computer while it is running.

7. System models

In this section we provide the models of our Gazillion game system by means of different UML diagrams: use case, sequence, state and activity; as well as through user interface mock-ups.

7.1 Use case model

Gazillion game consists of one actor (Player), whose interactions with the system are depicted in the Use Case Diagrams below. For each action in the first diagram, the flow of events and entry/exit conditions are explained in detail in the tables below.

The first diagram is a general use case diagram for the menu interactions between the player and the system. The player is able to choose among options to change the settings, read the instructions for the game, enter the shop, and play the game by choosing a game mode. The user may choose from three game modes, namely the Levels Mode, Treasure Mode, and the Ladder Mode. In each of these modes, the player may play a game of Quadrillion, in which the player may collect and use powerups, collect coins, and may time out if the player isn't able to finish the game in time. The player may also enter a shop from the main menu, from which the user can buy powerups and themes.

The second diagram is a use case diagram demonstrating the core gameplay aspects for a single game of Quadrillion. While playing a game of Quadrillion, the player is able to select a piece, rotate this selected piece, and flip this piece. The player may also place the selected piece on the game board, and remove a piece from the game board. In the case that the placement of the piece is invalid, the piece is not placed at all. In addition to these actions, the player may decide to give up, which ends the game, and the game counts as a loss. The player may use two power-ups during the course of the game, namely the health and the time power-ups. The player may also "reset" the game, which merely returns all the pieces back to their slots.

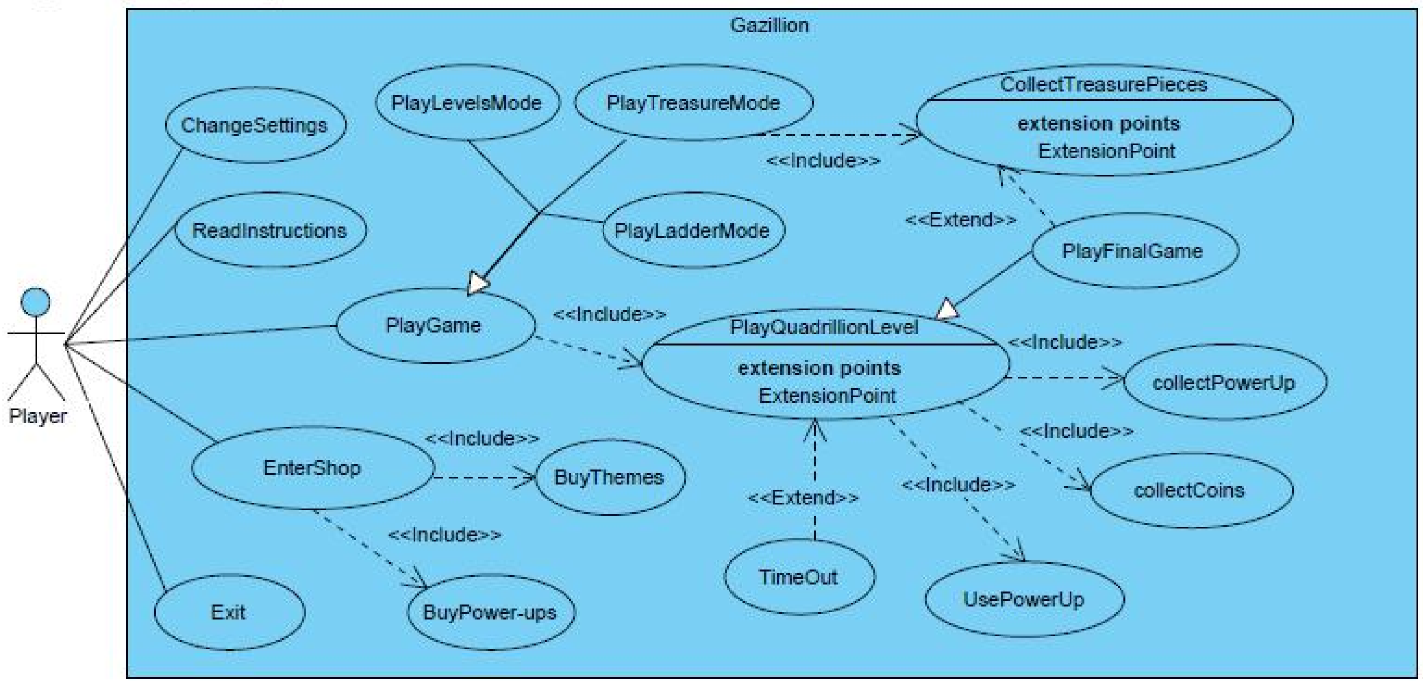


Figure 1. Gazillion Game Use Case Diagram

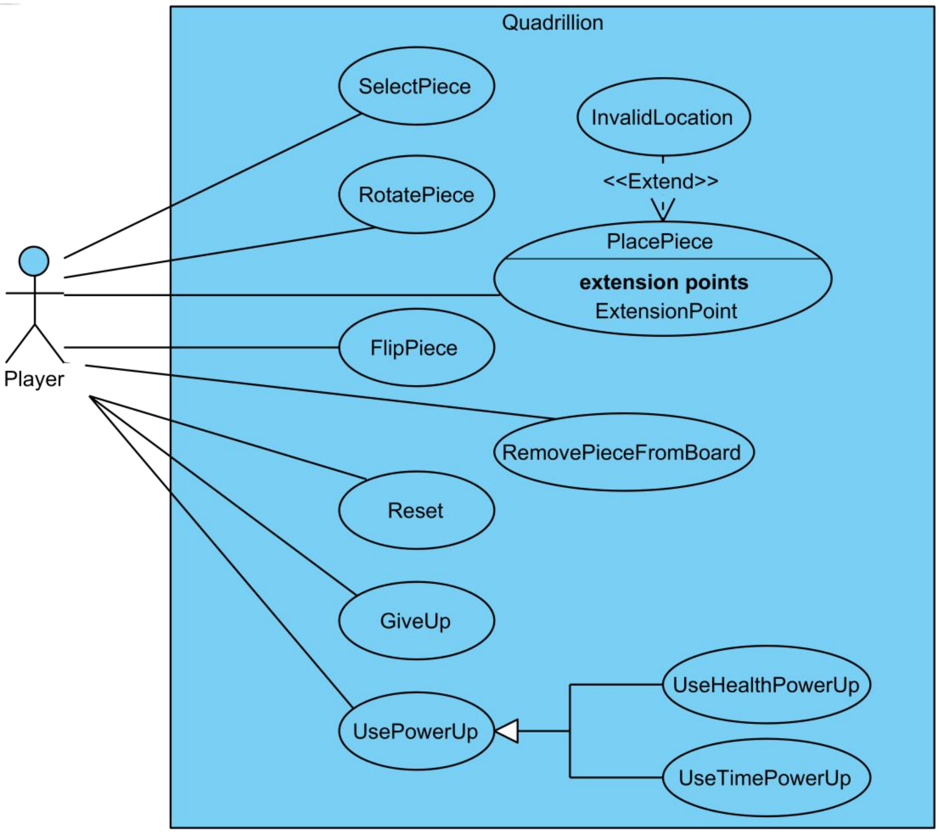


Figure 2. Quadrillion Game Use Case Diagram

USE CASE ReadInstructions	
Participating actors:	Player
Entry conditions:	Player is in the Main Menu and has pressed the Instructions button.
Exit conditions:	Gazillion provided Player with all the necessary information about the game rules, modes, objects and control keys.
Flow of events:	1. Player presses the Instructions button. 2. Game displays a window with the instructions. 3. Player scrolls down until he finds the part he is interested in and reads the instructions. 4. Player clicks on the Back button and returns to Main Menu.
Special requirements:	Player understands the language of the game.

USE CASE ChangeSettings	
Participating actors:	Player
Entry conditions:	Player is in the main menu and has pressed the Change Settings button.

Exit conditions:	• Configuration changes that the Player indicated have been applied. • Player returns to the main menu.
Flow of events:	1. Player presses the ChangeSettings button. 2. Game displays the settings options: Adjust volume, Change theme and Mute sound. A volume slider is and the available themes are also displayed. 3. Following events can happen in any order any number of times: 3.1 Player adjusts volume using the volume slider. 3.2 Player clicks on a theme to activate it. 3.3 Player mutes system sound by clicking on the corresponding button. 4. The game applies player's preferences immediately. 5. Player clicks on Back button to return to the Main Menu.
Special requirements:	Game must apply the changes specified by Player immediately.

USE CASE EnterShop	
Participating actors:	Player
Entry conditions:	• Player has pressed the Shop button in the main menu. • Player has enough coins to make the purchase.
Exit conditions:	• Player has the items he wanted to purchase. • Player can use them during the game in the suitable modes.
Flow of events:	1. Player presses the Shop button. 2. A window pops up and game provides player with 3 possibilities: Buy a hint, Buy a health potion and Buy extra time. *The options are displayed as buttons. Their respective prices are also shown.* 3. Player clicks on a button to make a purchase. 4. The system checks if Player has enough coins. If so, system updates Player information to include the bought item/power up. 5. The system subtracts the correct amount from player's coins (0 is no purchase is made). 6. Steps 2-4 repeat until Player presses the Back button to go back to the previous state.
Special requirements:	Player's inventory and budget should be immediately updated when a purchase is made.

USE CASE Exit	
Participating actors:	Player
Entry conditions:	• Player wants to quit the game. • Player has pressed the Exit button in the Main Menu.
Exit conditions:	• Game state is not saved after Player leaves the game (ex. score, coins, power ups, level progress).
Flow of events:	1. Player presses the Exit button. 2. Game warns Player that the progress will be lost and prompts the Player to confirm the action. 3. Player confirms he wants to exit the game. 4. Game ends and game window closes.
Alternative Flow:	1. Player confirms he does not want to exit the game by pressing Close. 2. Game redirects Player to Main Menu.
Special requirements:	When a new game is opened after a previous game was closed, it starts from the beginning.

USE CASE PlayQuadrillionLevel	
Participating actors:	Player
Entry conditions:	• Player is in LevelMode and has chosen a level to play OR • Player is in TreasureMode and has chosen a level on the map OR • Player is in LadderMode.
Exit conditions:	• Player receives an award if he won the game (depending on the game mode) or his health is decreased if he lost in Treasure Mode. • Any power up that the Player used during the game is removed from his inventory. • Player is redirected to the previous window he was in before the level started.
Flow of events:	1. System displays a window with a game on it. 2. Player picks up a quadrillion piece and flips or rotates it in any position using the corresponding control keys. 3. Player tries to place the piece in the grid. 4. Game checks if the piece can be placed where the Player indicated. 4.1 If yes, Game places the piece in the grid. 4.2 If not, Game puts the piece back in its place outside the grid. 5. Player can use any power up available in his inventory at any

	time (type of power ups depends on the game mode). 6. Steps 2-5 are repeated until the game ends: 6.1 Player loses the game if time runs out in a timed level. Player also loses if he clicks on Quit Game. 6.2 Player wins the game if all the pieces are placed on the grid on time. 7. Game updates Player information depending on the outcome: 7.1 Game awards Player if he won the game. Awards can be coins, power ups or puzzle pieces depending on the game mode. 7.2 Game decreases Player health by one unit if Player lost the level in Treasure Mode. 8. Player loses the Gazillion game if his health reaches 0. Otherwise he is redirected to the previous state: the window of the mode he was playing before the level started.
Special requirements:	Game must make sure that Player will not receive any award again by replaying the same level.

USE CASE PlayTreasureMode	
Participating actors:	Player
Entry conditions:	Player clicked on Play Game in the Main Menu and then Play Treasure Mode.
Exit conditions:	• Game saves the state and Player progress after Player goes back to the Main Menu. • Player inventory, budget and health remain the same as just before Player has exited Treasure Mode, regardless of which mode Player enters next.
Flow of events:	1. Player selects Play Treasure mode option from Main Menu. 2. Game displays a treasure map with levels all over it. Initially only the levels at the corner are available for Player to play. 3. Player chooses a level from the available ones. 4. Game initiates *timed* PlayQuadrillionLevel Use Case. 5. Player information is updated according to last level's outcome: 5.1 Player loses one unit of health if he loses the level. If the health reaches 0 Player loses the game completely. 5.2 If Player wins the level he gets awarded: any Power Up, coins or a puzzle piece (treasure) for the "game inside the game". 6. If player won last level, Game unlocks the adjacent levels in the map.

	7. Player can activate hints to get information about the location of the puzzle pieces for the “game inside the game” in the treasure map. 8. Game displays a message with direction information. 9. Player can check the puzzle pieces he already found by clicking on the Treasure Inventory button. 10. Game displays a window with a Gazillion level on it, in which only the pieces that the Player has found are shown. 11. Player can place the pieces inside that grid just like in a normal level. Player wins Gazillion if he can complete this special level. 12. Game always records the changes Player makes in this grid. 13. Player clicks exit to go back to the Main Menu. 14. Game saves the state of the Treasure Mode and Player information.
Special requirements:	None

USE CASE PlayLevelsMode	
Participating actors:	Player
Entry conditions:	Player clicked on Play Game in the Main Menu and then Play Levels Mode.
Exit conditions:	• Game saves the state and Player progress after Player goes back to the Main Menu. • Player inventory, budget and health remain the same as just before Player has exited Levels Mode, regardless of which mode Player enters next.
Flow of events:	1. Player selects Play Levels Mode option from Main Menu. 2. Game displays a map with numbered levels all over it, labelled according to difficulty. Initially only the first level is available. 3. Player chooses the level he wants to play. 4. Game initiates *untimed* PlayQuadrillionLevel use case. 5. Player information is updated according to last level’s outcome: 5.1 If Player loses a level he does *not* lose health. 5.2 If Player wins the level he gets awarded either a Power Up or coins. 6. If Player won last level, Game makes the next level available to the Player if it was previously locked. 7. Player clicks on Back to Main Menu to exit LevelsMode. 8. Game saves Player information and his progress in Levels Mode.
Special	Player may not use any Power up in this mode and health is not

requirements:	affected.

USE CASE PlayLadderMode	
Participating actors:	Player
Entry conditions:	Player clicked on Play Game in the Main Menu and then Play Ladder Mode.
Exit conditions:	Player inventory, budget and health remain the same as just before Player has exited Ladder Mode, regardless of which mode Player enters next.
Flow of events:	1. Player selects Play Ladder Mode option from Main Menu. The overall time he has to play Ladder Mode is initialized to a predefined amount. 2. Game randomly initiates untimed PlayQuadrillionGame use case. 3. Player information is updated according to last level's outcome: 3.1 If time runs out Ladder Mode is over. Player is shown his final score (number of levels he passed in the time limit.) 3.2 If Player wins the level he gets awarded coins. Game initiates the next *untimed* PlayQuadrillionLevel use case directly. 4. Steps 2-3 are repeated until time runs out. 5. Player clicks on Back to Main Menu to exit LadderMode. 6. Game saves Player information but not Player progress.
Special requirements:	• Every time Player enters this mode time and score are reset. • Player may not use any Power up in this mode and health is not affected.

7.2 Object and class model

This is the class diagram of the Gazillion game. The whole diagram is divided into parts and displayed below. The link to the online version of the diagram in high resolution is https://github.com/annapecini/quadrillion/blob/unstable/doc/class_diagram.png

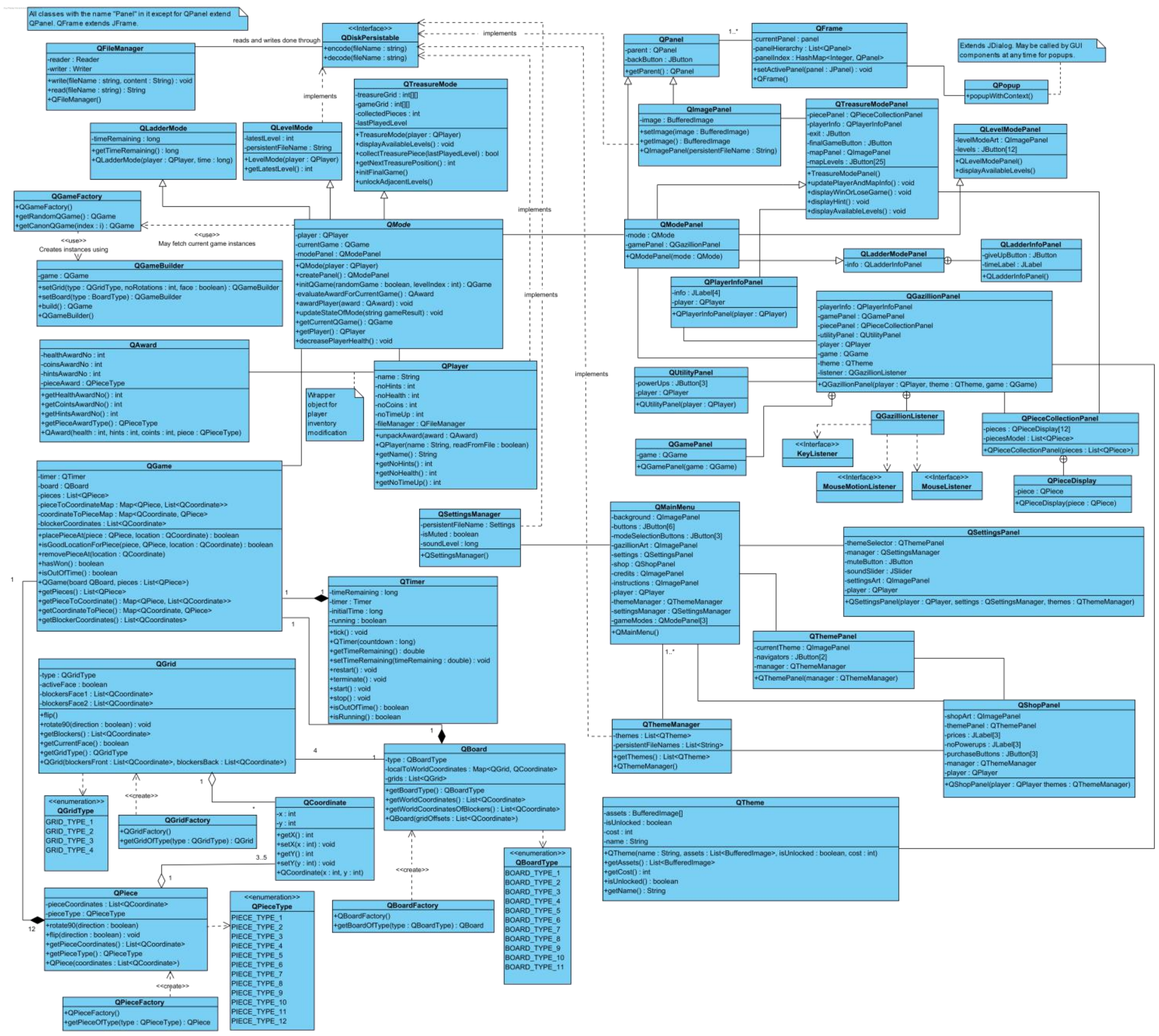


Figure 3. Class Diagram

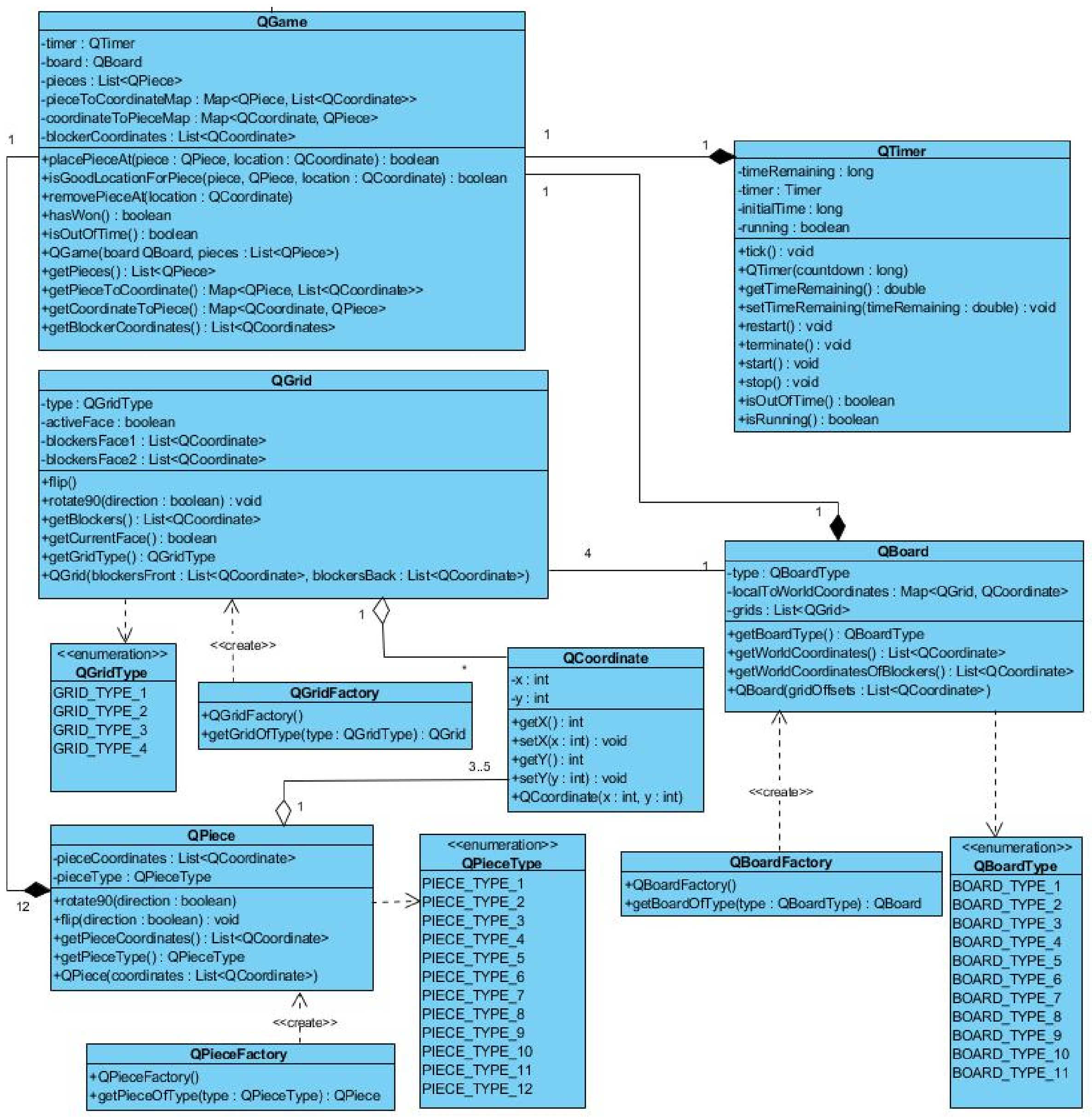

QGame
-timer : QTimer
-board : QBoard
-pieces : List<QPiece>
-pieceToCoordinateMap : Map<QPiece, List<QCoordinate>>
-coordinateToPieceMap : Map<QCoordinate, QPiece>
-blockerCoordinates : List<QCoordinate>
+placePieceAt(piece : QPiece, location : QCoordinate) : boolean
+isGoodLocationForPiece(piece, QPiece, location : QCoordinate) : boolean
+removePieceAt(location : QCoordinate)
+hasWon() : boolean
+isOutOfTime() : boolean
+QGame(board QBoard, pieces : List<QPiece>)
+getPieces() : List<QPiece>
+getPieceToCoordinate() : Map<QPiece, List<QCoordinate>>
+getCoordinateToPiece() : Map<QCoordinate, QPiece>
+getBlockerCoordinates() : List<QCoordinates>
QTimer
-timeRemaining : long
-timer : Timer
-initialTime : long
-running : boolean
+tick() : void
+QTimer(countdown : long)
+getTimeRemaining() : double
+setTimeRemaining(timeRemaining : double) : void
+restart() : void
+terminate() : void
+start() : void
+stop() : void
+isOutOfTime() : boolean
+isRunning() : boolean
QGrid
-type : QGridType
-activeFace : boolean
-blockersFace1 : List<QCoordinate>
-blockersFace2 : List<QCoordinate>
+flip()
+rotate90(direction : boolean) : void
+getBlockers() : List<QCoordinate>
+getCurrentFace() : boolean
+getGridType() : QGridType
+QGrid(blockersFront : List<QCoordinate>, blockersBack : List<QCoordinate>)
QBoard
-type : QBoardType
-localToWorldCoordinates : Map<QGrid, QCoordinate>
-grids : List<QGrid>
+getBoardType() : QBoardType
+getWorldCoordinates() : List<QCoordinate>
+getWorldCoordinatesOfBlockers() : List<QCoordinate>
+QBoard(gridOffsets : List<QCoordinate>)
<<enumeration>>
QGridType
GRID_TYPE_1
GRID_TYPE_2
GRID_TYPE_3
GRID_TYPE_4
<<create>>
QGridFactory
+QGridFactory()
+getGridOfType(type : QGridType) : QGrid
QCoordinate
-x : int
-y : int
+getX() : int
+setX(x : int) : void
+getY() : int
+setY(y : int) : void
+QCoordinate(x : int, y : int)
<<create>>
QPiece
-pieceCoordinates : List<QCoordinate>
-pieceType : QPieceType
+rotate90(direction : boolean)
+flip(direction : boolean) : void
+getPieceCoordinates() : List<QCoordinate>
+getPieceType() : QPieceType
+QPiece(coordinates : List<QCoordinate>)
<<enumeration>>
QPieceType
PIECE_TYPE_1
PIECE_TYPE_2
PIECE_TYPE_3
PIECE_TYPE_4
PIECE_TYPE_5
PIECE_TYPE_6
PIECE_TYPE_7
PIECE_TYPE_8
PIECE_TYPE_9
PIECE_TYPE_10
PIECE_TYPE_11
PIECE_TYPE_12
QBoardFactory
+QBoardFactory()
+getBoardOfType(type : QBoardType) : QBoard
<<enumeration>>
QBoardType
BOARD_TYPE_1
BOARD_TYPE_2
BOARD_TYPE_3
BOARD_TYPE_4
BOARD_TYPE_5
BOARD_TYPE_6
BOARD_TYPE_7
BOARD_TYPE_8
BOARD_TYPE_9
BOARD_TYPE_10
BOARD_TYPE_11
<<create>>
QPieceFactory
+QPieceFactory()
+getPieceOfType(type : QPieceType) : QPiece
1
1
1
1
1
4
1
1
*
3..5
1
12

Figure 4. Class Diagram for Game Logic

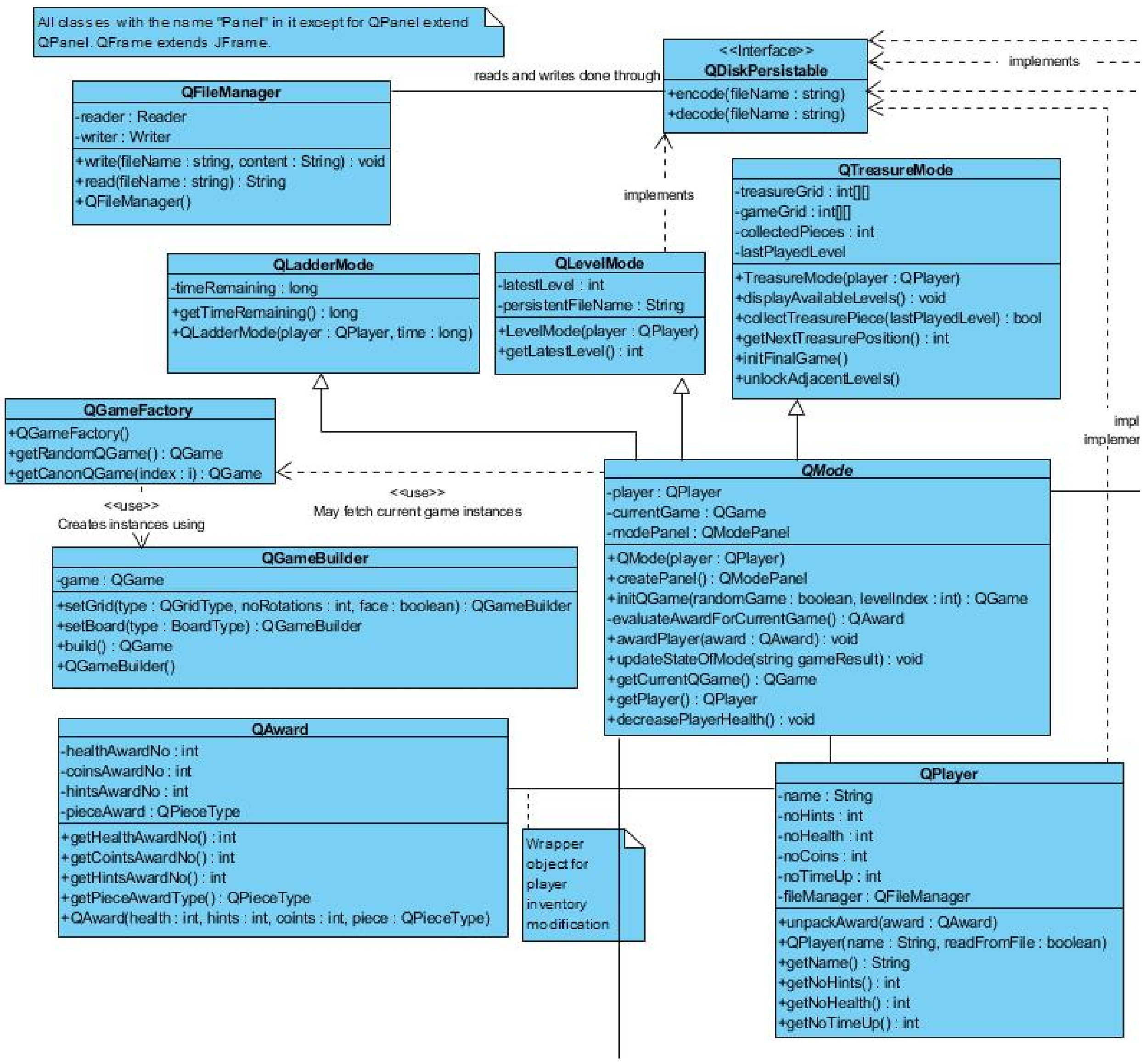

All classes with the name "Panel" in it except for QPanel extend QPanel. QFrame extends JFrame.
<<Interface>>
QDiskPersistable
+encode(fileName : string)
+decode(fileName : string)
implements
reads and writes done through
QFileManager
-reader : Reader
-writer : Writer
+write(fileName : string, content : String) : void
+read(fileName : string) : String
+QFileManager()
implements
QTreasureMode
-treasureGrid : int[][]
-gameGrid : int[][]
-collectedPieces : int
-lastPlayedLevel
+TreasureMode(player : QPlayer)
+displayAvailableLevels() : void
+collectTreasurePiece(lastPlayedLevel) : bool
+getNextTreasurePosition() : int
+initFinalGame()
+unlockAdjacentLevels()
QLadderMode
-timeRemaining : long
+getTimeRemaining() : long
+QLadderMode(player : QPlayer, time : long)
QLevelMode
-latestLevel : int
-persistentFileName : String
+LevelMode(player : QPlayer)
+getLatestLevel() : int
impl
implemer
QGameFactory
+QGameFactory()
+getRandomQGame() : QGame
+getCanonQGame(index : i) : QGame
<<use>>
Creates instances using
<<use>>
May fetch current game instances
QMode
-player : QPlayer
-currentGame : QGame
-modePanel : QModePanel
+QMode(player : QPlayer)
+createPanel() : QModePanel
+initQGame(randomGame : boolean, levelIndex : int) : QGame
-evaluateAwardForCurrentGame() : QAward
+awardPlayer(award : QAward) : void
+updateStateOfMode(string gameResult) : void
+getCurrentQGame() : QGame
+getPlayer() : QPlayer
+decreasePlayerHealth() : void
QGameBuilder
-game : QGame
+setGrid(type : QGridType, noRotations : int, face : boolean) : QGameBuilder
+setBoard(type : BoardType) : QGameBuilder
+build() : QGame
+QGameBuilder()
QAward
-healthAwardNo : int
-coinsAwardNo : int
-hintsAwardNo : int
-pieceAward : QPieceType
+getHealthAwardNo() : int
+getCointsAwardNo() : int
+getHintsAwardNo() : int
+getPieceAwardType() : QPieceType
+QAward(health : int, hints : int, coints : int, piece : QPieceType)
Wrapper object for player inventory modification
QPlayer
-name : String
-noHints : int
-noHealth : int
-noCoins : int
-noTimeUp : int
-fileManager : QFileManager
+unpackAward(award : QAward)
+QPlayer(name : String, readFromFile : boolean)
+getName() : String
+getNoHints() : int
+getNoHealth() : int
+getNoTimeUp() : int

Figure 5. Class Diagram for Modes

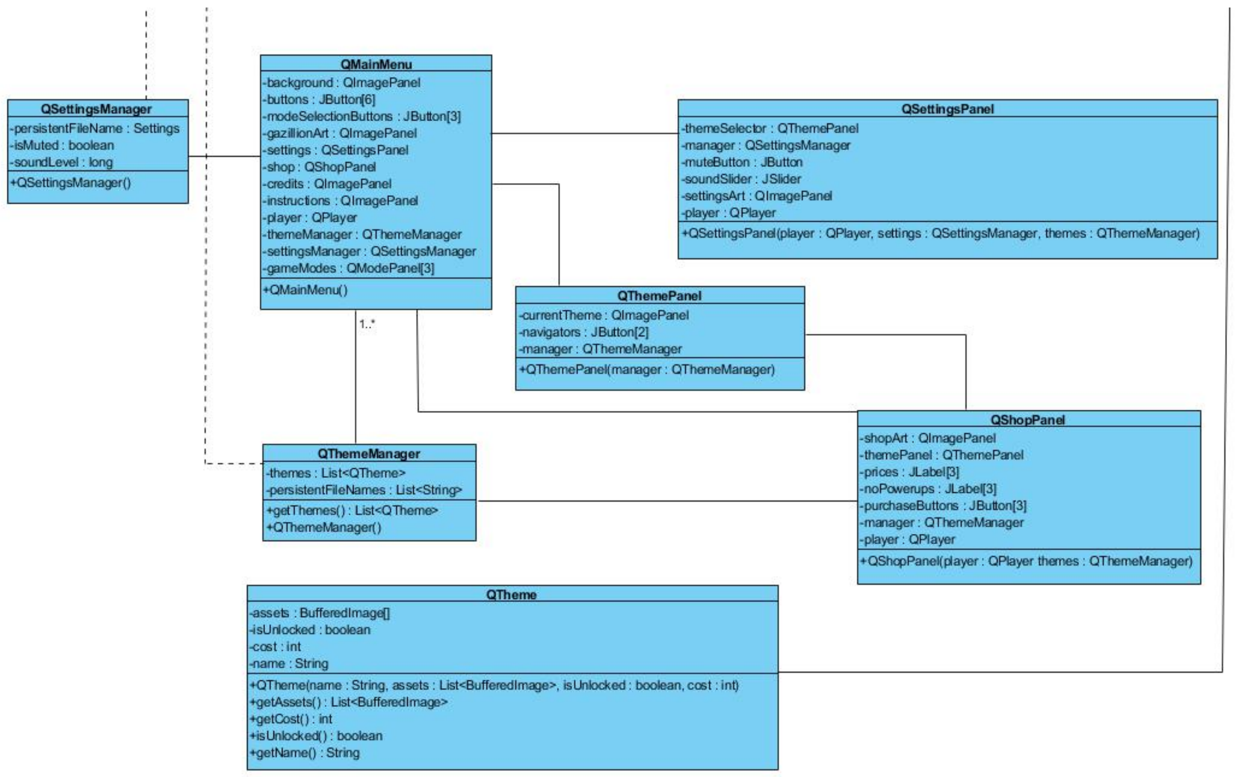

QSettingsManager
-persistentFileName : Settings
-isMuted : boolean
-soundLevel : long
+QSettingsManager()
QMainMenu
-background : QImagePanel
-buttons : JButton[6]
-modeSelectionButtons : JButton[3]
-gazillionArt : QImagePanel
-settings : QSettingsPanel
-shop : QShopPanel
-credits : QImagePanel
-instructions : QImagePanel
-player : QPlayer
-themeManager : QThemeManager
-settingsManager : QSettingsManager
-gameModes : QModePanel[3]
+QMainMenu()
1..*
QSettingsPanel
-themeSelector : QThemePanel
-manager : QSettingsManager
-muteButton : JButton
-soundSlider : JSlider
-settingsArt : QImagePanel
-player : QPlayer
+QSettingsPanel(player : QPlayer, settings : QSettingsManager, themes : QThemeManager)
QThemePanel
-currentTheme : QImagePanel
-navigators : JButton[2]
-manager : QThemeManager
+QThemePanel(manager : QThemeManager)
QShopPanel
-shopArt : QImagePanel
-themePanel : QThemePanel
-prices : JLabel[3]
-noPowerups : JLabel[3]
-purchaseButtons : JButton[3]
-manager : QThemeManager
-player : QPlayer
+QShopPanel(player : QPlayer themes : QThemeManager)
QThemeManager
-themes : List<QTheme>
-persistentFileNames : List<String>
+getThemes() : List<QTheme>
+QThemeManager()
QTheme
-assets : BufferedImage[]
-isUnlocked : boolean
-cost : int
-name : String
+QTheme(name : String, assets : List<BufferedImage>, isUnlocked : boolean, cost : int)
+getAssets() : List<BufferedImage>
+getCost() : int
+isUnlocked() : boolean
+getName() : String

Figure 6. Class Diagram for Menu

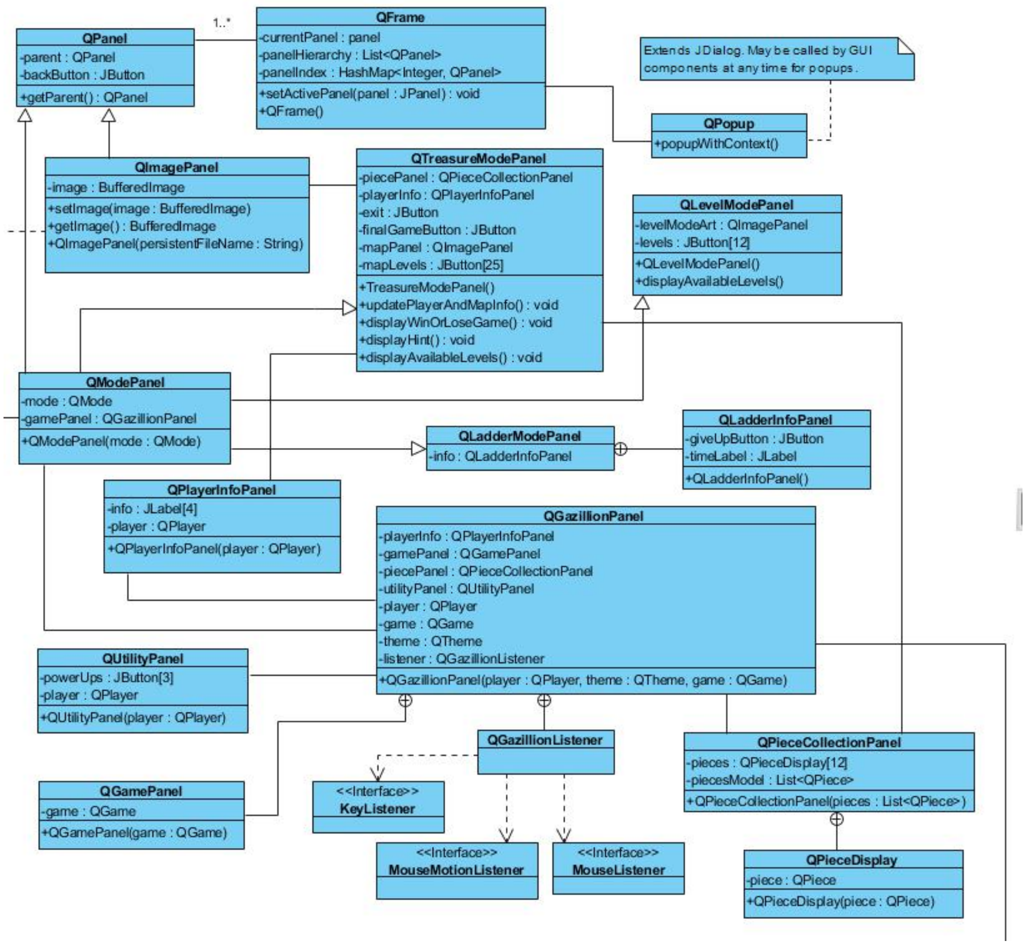

QPanel
-parent : QPanel
-backButton : JButton
+getParent() : QPanel
1..*
QFrame
-currentPanel : panel
-panelHierarchy : List<QPanel>
-panelIndex : HashMap<Integer, QPanel>
+setActivePanel(panel : JPanel) : void
+QFrame()
Extends JDialog. May be called by GUI components at any time for popups.
QPopup
+popupWithContext()
QImagePanel
-image : BufferedImage
+setImage(image : BufferedImage)
+getImage() : BufferedImage
+QImagePanel(persistentFileName : String)
QTreasureModePanel
-piecePanel : QPieceCollectionPanel
-playerInfo : QPlayerInfoPanel
-exit : JButton
-finalGameButton : JButton
-mapPanel : QImagePanel
-mapLevels : JButton[25]
+TreasureModePanel()
+updatePlayerAndMapInfo() : void
+displayWinOrLoseGame() : void
+displayHint() : void
+displayAvailableLevels() : void
QLevelModePanel
-levelModeArt : QImagePanel
-levels : JButton[12]
+QLevelModePanel()
+displayAvailableLevels()
QModePanel
-mode : QMode
-gamePanel : QGazillionPanel
+QModePanel(mode : QMode)
QLadderModePanel
-info : QLadderInfoPanel
QLadderInfoPanel
-giveUpButton : JButton
-timeLabel : JLabel
+QLadderInfoPanel()
QPlayerInfoPanel
-info : JLabel[4]
-player : QPlayer
+QPlayerInfoPanel(player : QPlayer)
QGazillionPanel
-playerInfo : QPlayerInfoPanel
-gamePanel : QGamePanel
-piecePanel : QPieceCollectionPanel
-utilityPanel : QUtilityPanel
-player : QPlayer
-game : QGame
-theme : QTheme
-listener : QGazillionListener
+QGazillionPanel(player : QPlayer, theme : QTheme, game : QGame)
QUtilityPanel
-powerUps : JButton[3]
-player : QPlayer
+QUtilityPanel(player : QPlayer)
QGazillionListener
QPieceCollectionPanel
-pieces : QPieceDisplay[12]
-piecesModel : List<QPiece>
+QPieceCollectionPanel(pieces : List<QPiece>)
QGamePanel
-game : QGame
+QGamePanel(game : QGame)
<<Interface>>
KeyListener
<<Interface>>
MouseMotionListener
<<Interface>>
MouseListener
QPieceDisplay
-piece : QPiece
+QPieceDisplay(piece : QPiece)

Figure 7. Class Diagram for GUI

- MainMenu

MainMenu is the main interface of our program from where the player can choose the mode of the game (Treasure mode, Ladder mode, Levels mode), can go to the Shop, to Settings, to Instructions page or Credits page, start the game with the chosen settings passed as parameters and exit the application. MainMenu creates an instance of QMode, an instance of Shop and an instance of Settings.

- Settings

Settings will allow the player to change themes and volume level. Settings class has one attribute of integer type which represents the index of the chosen theme stored in an array of themes. The theme is changed by simply changing the index.

- Shop

Shop class will have a list of themes which will initially be locked and an instance of the Player. As the player progresses and gets rewards, the Player can purchase more themes. The player can also buy power ups. The method buyPowerUps() increases the player's hintNumber, timePowerUp or healthPowerUp according to his choice. This power ups can be used later during the game.

- Theme

A theme is represented by an attribute of type BufferedImage and an attribute of type boolean which determines whether the theme is locked or not. The attributes also have the corresponding setters and getters.

- Player

A player has attributes that indicate his number of remaining life and total amount of coins. Player has also the attributes that represent the number of power ups he has gathered throughout the game but has not used yet. This class has a list of instances of the PowerUp interface, by which it can call the method to use the available power ups.

- Hint, Time, Health

These classes implement the PowerUp interface whose main method is usePowerUps(). In Hint subclass, this method returns a message according to the position of the player in the map (Hints are available only on Treasure mode) with respect to the nearest treasure piece.

In Time subclass, this method increases time by an amount of timeBonus and in Health subclass, the method increases player's healthNumber by a unit amount of bonusHealth.

- QMode

QMode is an abstraction for the game modes as described in the previous section. It has references to the player (QPlayer), the current game of Quadrillion that is being played by the player, and a reference to the panel on which the QMode object draws its specified graphics. The QMode abstraction provides a method that instantiates the panel corresponding to the QMode instance, which provides a graphical representation of the mode to the player. It also provides methods for awarding the player, evaluating the award for the game of Quadrillion, a method that updates the logical state of the mode, and a method that decreases the health of the player. It also requires that its child classes implement the initQGame() method, which allows the QGame object to be instantiated. Any mode that has to be logically registered to the game, has to extend this class and provide extra functionalities.

- QTreasureMode

The map in the treasure mode is represented by a double array of integers, where 1 are the visited entries by the player and 0 are the rest. Another double array of integers keeps track of the treasure positions and the state (collected, hidden, revealed). When player chooses a level by clicking on a specific position of a map, the position is translated into matrix entries and if the level is unlocked the gameGrid is updated and a new QGame is instantiated. The method displayPieces returns a list of QPiece type with all the pieces collected by the player so far. getNextTreasurePosition is called when the player uses hint power ups and TreasureMode passes the player position as a parameter to generate a relevant hint. If the player has collected all 12 pieces, he can click to play the final game, which again calls a QGame. If the player wins the final game, the treasure mode progress is restarted.

- QLadderMode

The Ladder mode keeps track of the current score and the time remaining in two distinct attributes, with their respective getter and setter methods.

- QLevelMode

In Levels mode, we store the highest available level so far and we continually update the level according to the player's progress.

- QCoordinate:

QCoordinate is a class which defines a two-tuple representing a coordinate on a discrete grid (i.e. a grid with only integer coordinates), with the necessary getters and setters.

- QGrid:

QGrid defines one of the four Quadrillion grids, each in its local coordinate system. Since a grid can only be 4x4, it does not explicitly store any QCoordinate objects regarding to the topology of the grid. Rather, it only stores the places on which no piece may be placed (blockers), on both faces of the grid. The class provides methods for the rotation and flipping of the board, along with the necessary getters.

- QPiece:

QPiece defines a Quadrillion piece in a local coordinate system by explicitly storing the coordinates of the piece. It provides methods for flipping, and rotation, and the necessary getters.

- QBoard:

QBoard defines a full game board, which consists of the spatial combination of four QGrids. For each grid, the class stores the offset of the local coordinate system in a map, list of the grids, and the type of the topology of the board. It provides the necessary getters, and a method to "materialize" the entire grid by building a list of coordinates. To build the list of coordinates, the grid coordinates are explicitly generated and for every grid, the offset of that grid is added to all coordinate objects, and all of them are appended to the list. The class also provides a method for expressing the "blocker" locations in world coordinates.

- QTimer:

QTimer is a simple timer class which keeps track of time elapsed by subtracting a fixed amount from the time remaining at every call of the tick() method it provides. Has the necessary getters and setters. Runs on a separate thread.

- QGame:

  QGame defines a game of Gazillion. It consists of 12 QPiece and 1 QBoard instances. Keeps track of where the pieces are, and the piece hosted by a particular coordinate. Provides methods for piece manipulation via aggregation with QPiece and QBoard instances. Provides methods that check whether the game is considered to be complete, or time has run out.

Enumerations:

- QPieceType:

  Enumerates the types of pieces as given in the Quadrillion game manual. There are twelve distinct pieces.

- QGridType:

  Enumerates the types of grids as given in the Quadrillion game manual. There are four distinct grids.

- QBoardType:

  Enumerates the types of boards that can be created using four grids. There are eleven official positions, and they are guaranteed to have at least one solution.

Factory Classes:

- QPieceFactory:

  Constructs specific instances of QPiece by specifying coordinates according to the type (enumeration) of the QPiece, as present in the game.

- QGridFactory:

  Constructs specific instances of QGrid by specifying the coordinates of the "blocked" coordinates according to the type of the QGrid, as present in the game. A grid may be uniquely defined by the location of its blockers on each face.

- QBoardFactory:

  Constructs specific instances of QBoard by specifying the offsets of the QGrid instances to express them in "world" or "board" coordinates. There are eleven configurations for these offsets in the game manual.

7.3 Dynamic models

This section contains dynamic model diagrams (sequence, state and activity) for the major use cases described in the previous section.

7.3.1 Sequence Diagrams

These are the sequence diagrams that depict the flow of events of the main functionalities in our program:

- QGame (basic Gazillion Game)

When the player starts playing the game the QGame instantiates a QBoard and 12 Piece instances. If the game is in Treasure or Ladder mode, a QTimer instance is also created. During the game, the player chooses one piece type, which he can rotate or flip and then chooses a location in the board to locate the piece. The game first checks if the position is available (i.e. the entire piece fits inside the bounds of the board and it is not placed over a blocker). Only if the position is available, the player can place the piece, and the game updates the corresponding mappings between pieces and coordinates.

A player also can remove a piece from the board. The mappings are updated after each movement on the game. Meanwhile, the QTimer is running on the background and it continuously notifies the game about the remaining time. If the player chooses to use time power ups, the QTimer increases the remaining time. If the time is up or the player has won, the game is completed.

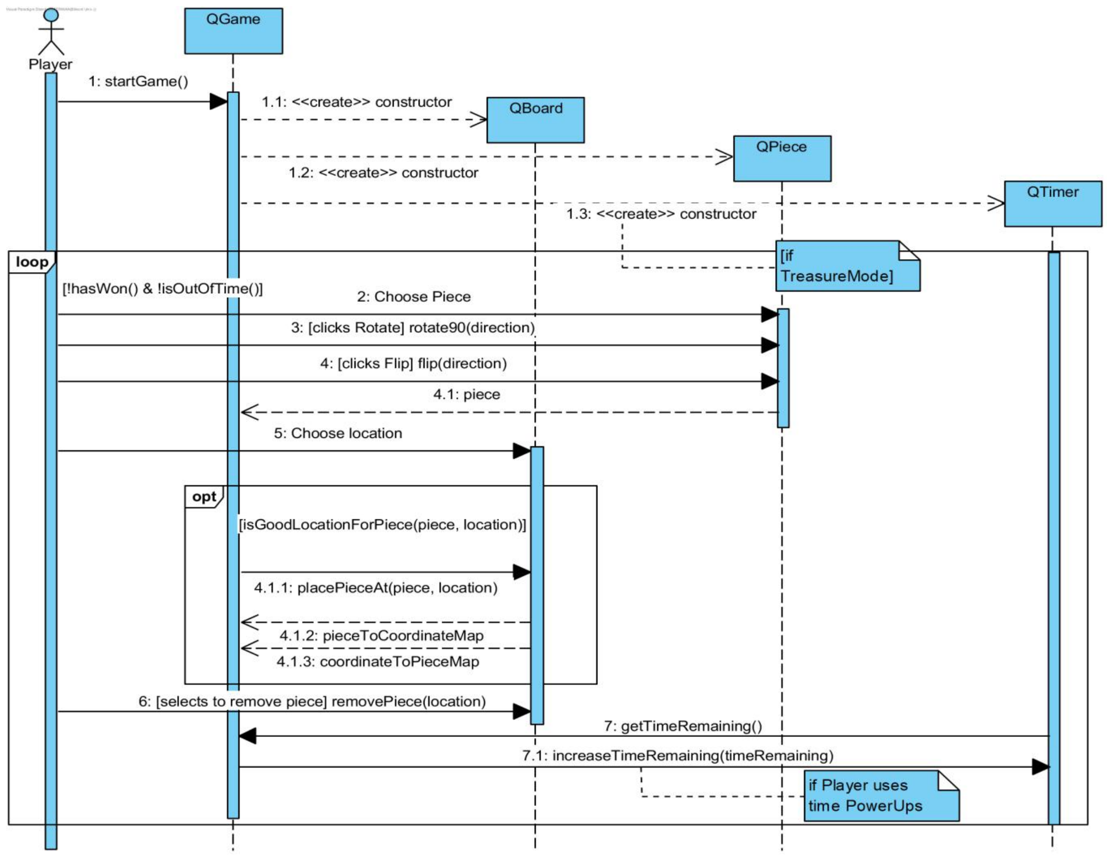

Player
QGame
QBoard
QPiece
QTimer
1: startGame()
1.1: <<create>> constructor
1.2: <<create>> constructor
1.3: <<create>> constructor
[if TreasureMode]
loop
[!hasWon() & !isOutOfTime()]
2: Choose Piece
3: [clicks Rotate] rotate90(direction)
4: [clicks Flip] flip(direction)
4.1: piece
5: Choose location
opt
[isGoodLocationForPiece(piece, location)]
4.1.1: placePieceAt(piece, location)
4.1.2: pieceToCoordinateMap
4.1.3: coordinateToPieceMap
6: [selects to remove piece] removePiece(location)
7: getTimeRemaining()
7.1: increaseTimeRemaining(timeRemaining)
if Player uses time PowerUps

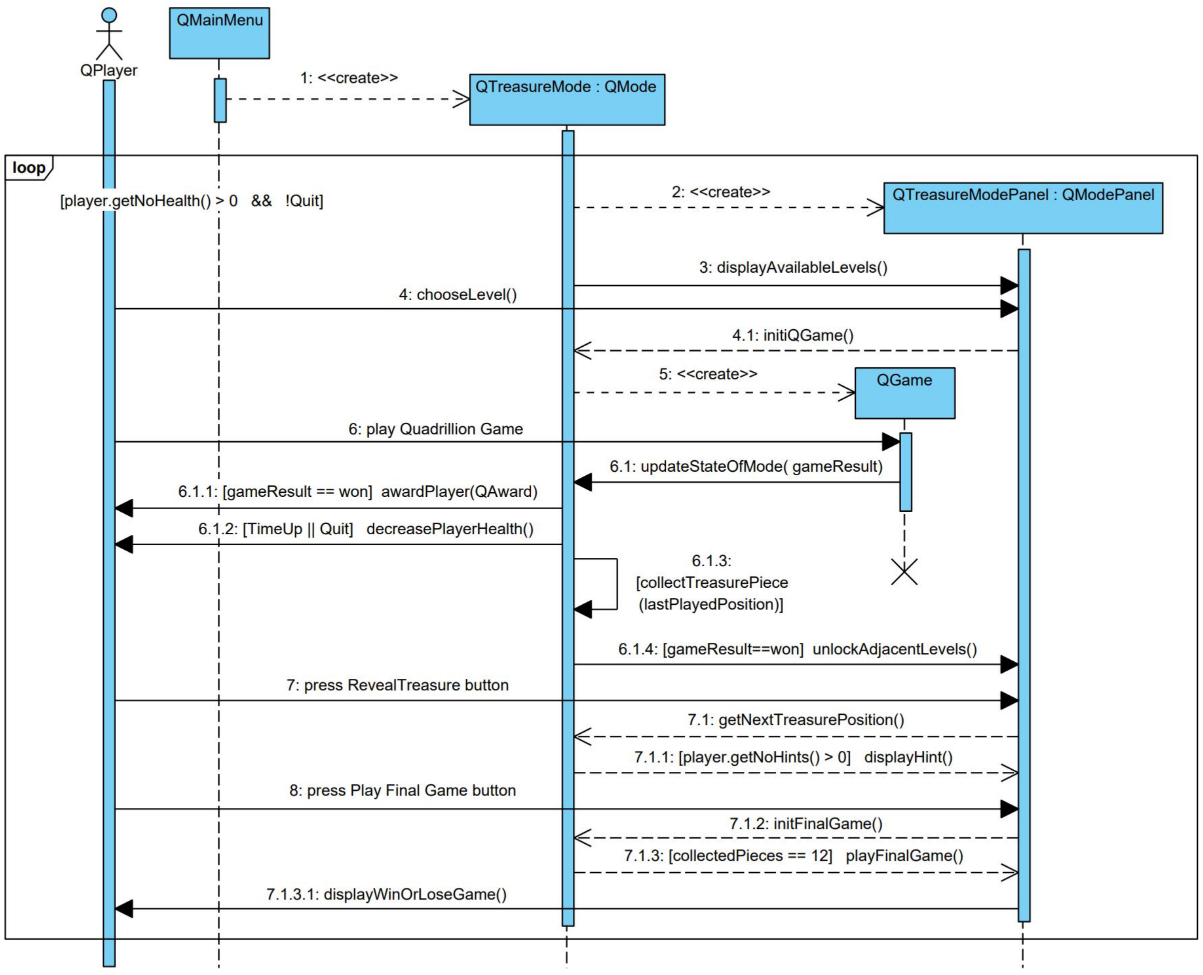
Figure 8. QGame Sequence Diagram

- QTreasureMode

After Player chooses the Treasure Mode from the Main Menu, the Treasure Mode Panel pops up and a treasure map (image) is displayed. Next to it a panel that contains all the collected treasure pieces is also displayed. There are locked levels all over the map that the Player can select to play. Initially only the corner levels are unlocked, and Player can unlock the adjacent levels by winning a game. While the Player's health is not 0 and Player has not played the Final game yet or pressed the Back Button to go to the Main Menu, he can select an unlocked level and play the random Quadrillion game generated by it. Player can use his power ups during the game, just as explained in the Quadrillion Game Sequence Diagram.

If Player wins, he gets awarded Power Ups in the form of health units, time bonuses or coins. If a treasure piece was hidden in that level/that part of the map, it is also "collected" by the Player and displayed in the Treasure Pieces panel next to the map. Player can choose which direction he would like to pursue on the treasure map (which level from the adjacent levels he would like to play next). If Player loses the level, a unit of health is subtracted. If Player presses Reveal Treasure, a hint about the next treasure position is displayed given the

Player has enough Treasure Hint power ups. The level "node" on which the treasure is found is indicated by changing the node colour. Only if he has gathered the 12 pieces can the Player play the Final Game, which is a normal Quadrillion game with the pieces Player has collected. He can win the ultimate Gazillion game only if he beats this last level.



QPlayer
QMainMenu
1: <<create>>
QTreasureMode : QMode
loop
[player.getNoHealth() > 0 && !Quit]
2: <<create>>
QTreasureModePanel : QModePanel
3: displayAvailableLevels()
4: chooseLevel()
4.1: initiQGame()
5: <<create>>
QGame
6: play Quadrillion Game
6.1: updateStateOfMode(gameResult)
6.1.1: [gameResult == won] awardPlayer(QAward)
6.1.2: [TimeUp || Quit] decreasePlayerHealth()
6.1.3: [collectTreasurePiece (lastPlayedPosition)]
6.1.4: [gameResult==won] unlockAdjacentLevels()
7: press RevealTreasure button
7.1: getNextTreasurePosition()
7.1.1: [player.getNoHints() > 0] displayHint()
8: press Play Final Game button
7.1.2: initFinalGame()
7.1.3: [collectedPieces == 12] playFinalGame()
7.1.3.1: displayWinOrLoseGame()

Figure 9. Treasure Mode Sequence Diagram

- QMode

The main flow of events of QLevelMode and QLadderMode is depicted on a single diagram below, because they are very similar in behavior. In these modes similar methods are called and executed but the function body is different for each of them.

When the player chooses one of these modes from the MainMenu, the corresponding child class of QMode is instantiated. Once the QMode is created, it activates the QModePanel. The game is performed until the player presses 'back' button or in the case of the ladder mode if the time is up. In the level mode the QLevelModePanel displays only the available levels, according to the logic variables stored in QMode. Then, he is asked to choose one level to play. The chosen level is passed as an input to the QMode to initialize the game. In the ladder mode, however the player does not choose the levels.

In game initialization QMode creates an instance of QGame. The player then plays Quadrillion Game which consists of a series of events described separately in the QGame Sequence Diagram (Figure 8). At the end of the game, state of the mode is updated and the award for the player (amount of coins, health, time power- ups) is evaluated according to the game result.

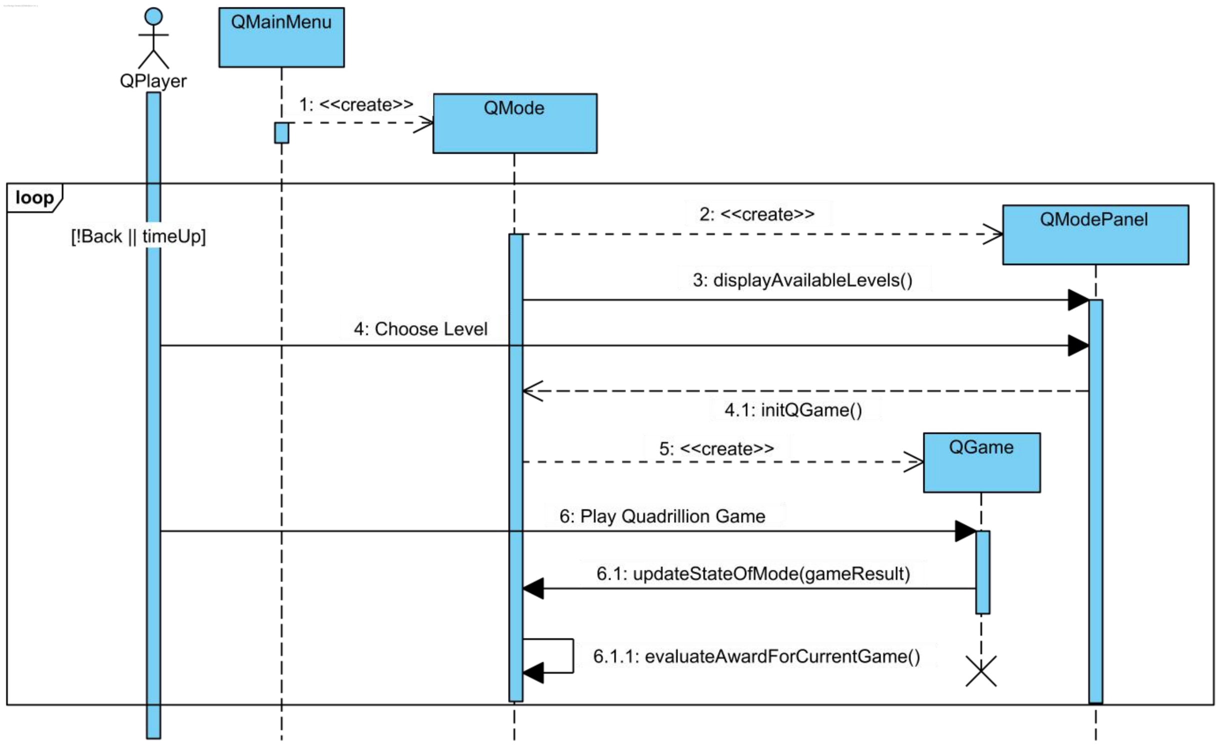


Figure 10. QMode Sequence Diagram

7.3.2 State Diagrams

Below are provided the state diagrams for each mode of Gazillion and QPiece object.

- QPiece (Gazillion piece)

The following is the state diagram for a QPiece object. When an instance of QGame is launched, the QPiece is, always, considered to be in the state of "Out of the board". If the Player clicks on a piece, anywhere, the QPiece state is set to "selected". When a QPiece is selected, the Player may rotate, flip, or attempt to place the QPiece object. Rotations and flips do not change alter the state of the QPiece, and the QPiece is considered to be selected after rotations and flips. In the case of an attempt in placement, if the Player fails to appropriately place the object on the board, the QPiece instance will be still “selected”. If placed appropriately, the QPiece transitions into a state of being on the board / in play. If the final end-game state has not yet been reached, if a QPiece on the board is selected by the Player via a mouse click, it is marked as Selected. The "meta" final condition for a QPiece object is when all pieces are in the state of "on the board", i.e. the puzzle has been completed.

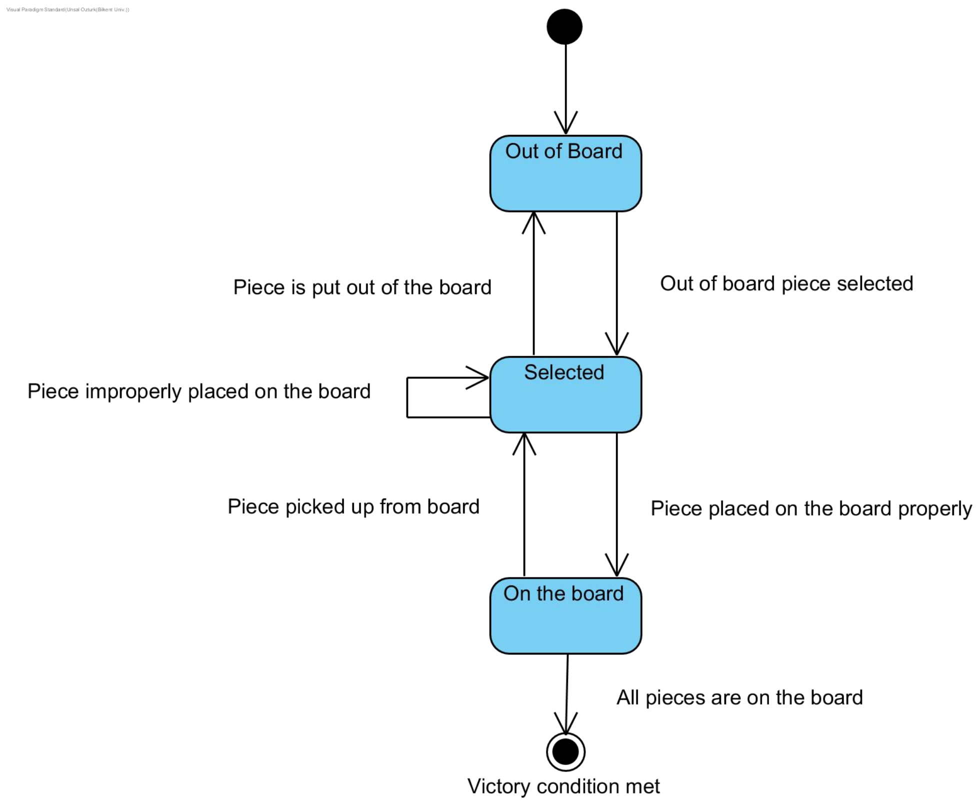

Out of Board
Piece is put out of the board
Out of board piece selected
Piece improperly placed on the board
Selected
Piece picked up from board
Piece placed on the board properly
On the board
All pieces are on the board
Victory condition met

Figure 11. QPiece State Diagram

- Treasure Mode

The following is the state diagram for the treasure mode. The treasure mode has four internal logical states. The map view state is the state of the system when the treasure mode is initially started up. In this state, the game displays a map from which several levels may be chosen from. When the player decides to play one of the levels in treasure mode, the game transitions into a state called Quadrillion Game State. In this state, the user plays a game of Quadrillion, and either wins or loses the game. When the game is over, the system enters a Mode Victory/Defeat and Award Evaluation state, in which the system decides on the award/punishment the player is subject to. In this state, the state of the progression in the mode is also evaluated. If the player has more than 0 health and has collected less than 12 pieces, the state is rolled back to the Map View State, from which the user may follow the same transitions all over again. If the user has 0 health remaining at the end of the evaluation state, the treasure mode terminates and the user is declared to be defeated. Otherwise, if the user has 12 pieces and more than 0 health, the treasure mode transitions into the Final Quadrillion Game State. If the user wins the final game, the user is declared to have won the treasure mode, otherwise the user loses - which are denoted by the final states of the treasure mode.

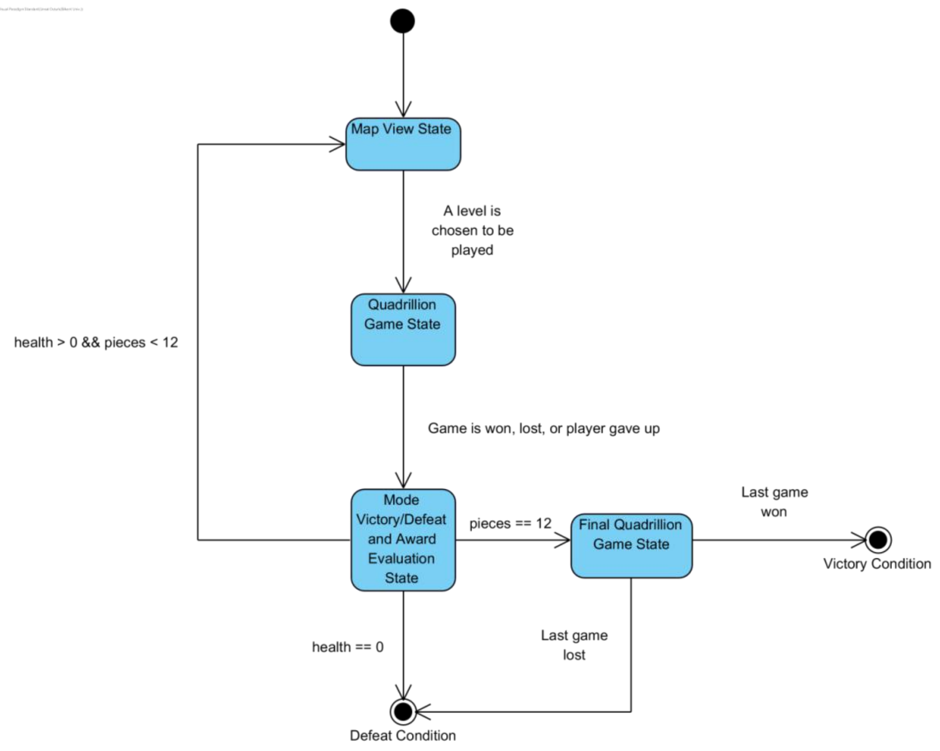

Map View State
A level is chosen to be played
Quadrillion Game State
health > 0 && pieces < 12
Game is won, lost, or player gave up
Mode Victory/Defeat and Award Evaluation State
pieces == 12
Final Quadrillion Game State
Last game won
Victory Condition
health == 0
Last game lost
Defeat Condition

Figure 12. Treasure Mode State Diagram

- Ladder Mode

The following is the state diagram for the Ladder mode. Upon starting the Ladder mode, the player is greeted by the game mode, and the game automatically transitions into a state of "Quadrillion Game". The state is retained until either the Player wins or runs out of time. If the Player wins, the score of the Player is incremented, and the Player is given an intermission between two levels of Quadrillion, as indicated by the Score Increment and Round Intermission state. The game loads another game of Quadrillion when this state is terminated internally by the game logic. Whenever the Player runs out of time in the state of Quadrillion game or decides to quit the game, the Player is displayed their final score, and when this display is dismissed, a final state of "Exit Ladder Mode" is achieved.

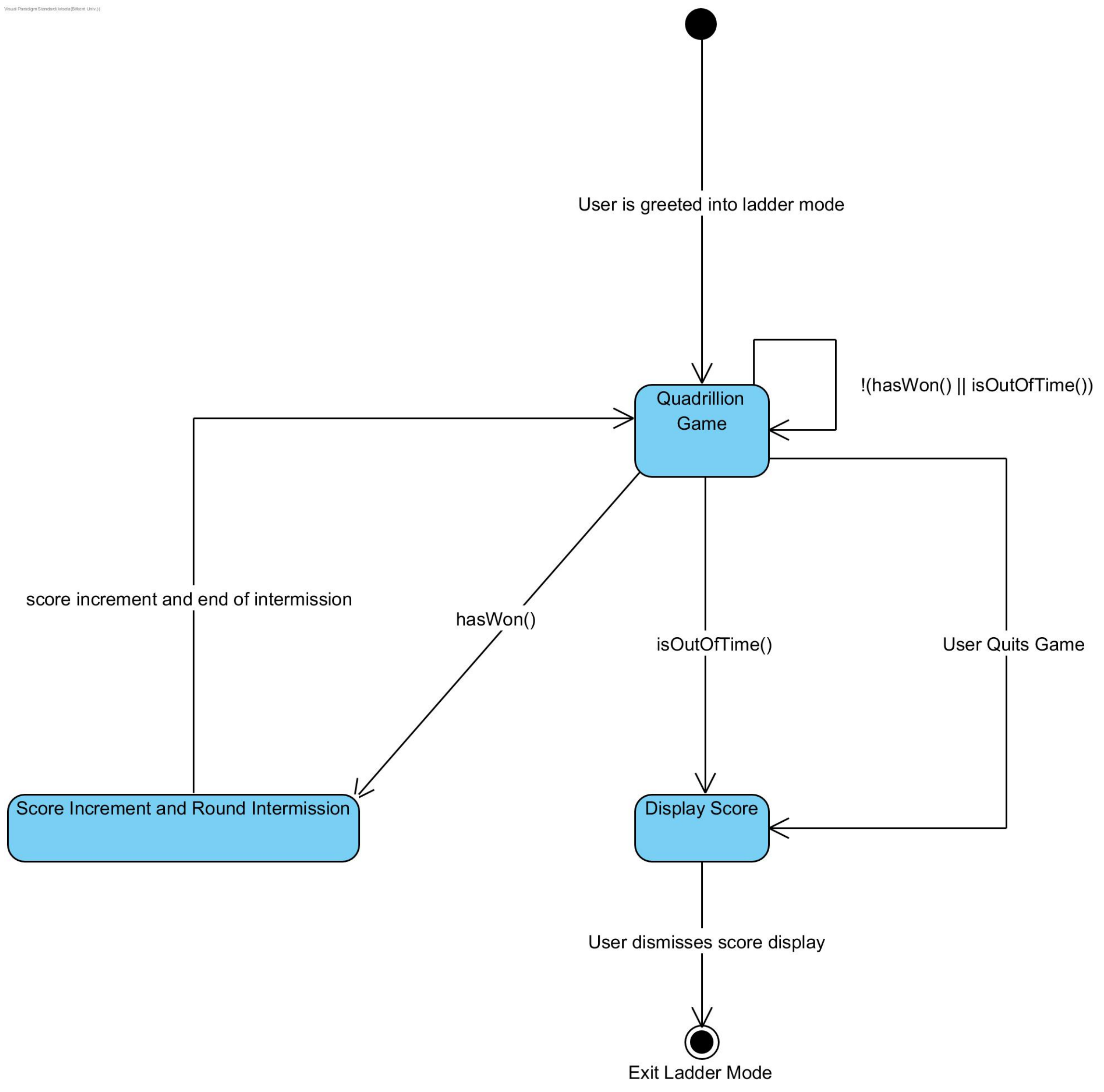

User is greeted into ladder mode
Quadrillion Game
!(hasWon() || isOutOfTime())
score increment and end of intermission
hasWon()
isOutOfTime()
User Quits Game
Score Increment and Round Intermission
Display Score
User dismisses score display
Exit Ladder Mode

Figure 13. Ladder Mode State Diagram

7.3.3 Activity Diagram

The following activity diagram describes the flow of mode selection, and player-mode dynamics. The system initially waits for the player to choose the game mode the player wishes to play. In the activity diagram this is denoted with the "Get user input for mode" activity. After the input is given to the system, the proper mode is loaded/instantiated. Then, after the player chooses a level, the system loads the level of choice. The player plays the game and finishes the round of Quadrillion, which is either a victory or a defeat. The system then gets the information about the gamer results, and then concurrently updates the state of the game mode and the player information. Upon finishing this update, the system, according to the logic of the game mode, checks whether the mode should end, or not. If the system deems that the game should not end, the system either lets the player continue playing the mode by allowing the player to choose another level, or terminates the current mode and lets the player choose another game mode. If the system thinks that the mode should end, the system ends the flow by directly terminating the game mode. In some cases, the mode has a final level and from this activity is initiated if the system logic allows it. From this activity, the flow ends in either a win or a loss, depending on whether the player won the final game.

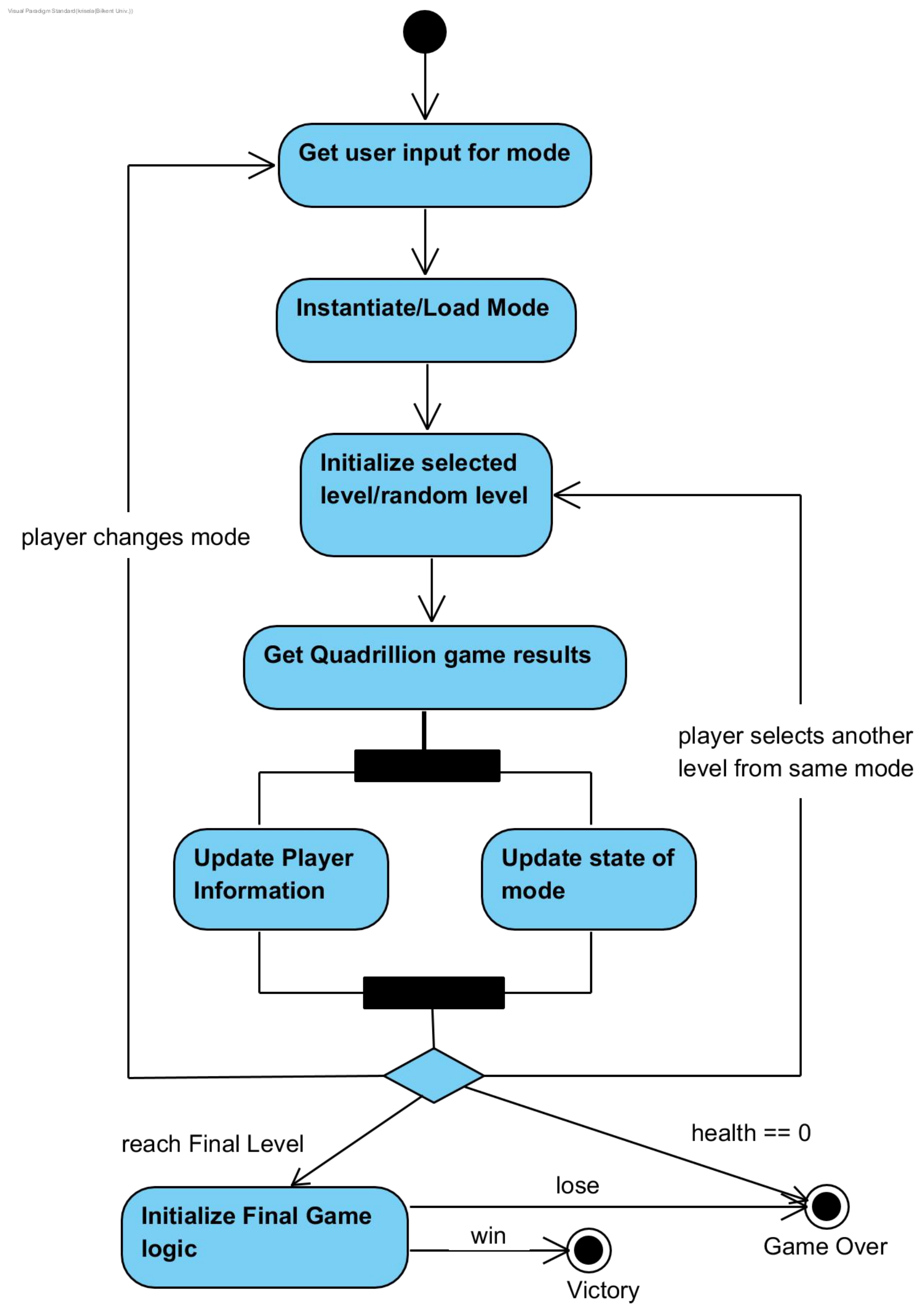

Visual Paradigm Standard(krisela(Bilkent Univ.))
Get user input for mode
Instantiate/Load Mode
Initialize selected level/random level
player changes mode
Get Quadrillion game results
player selects another level from same mode
Update Player Information
Update state of mode
reach Final Level
health == 0
lose
Initialize Final Game logic
win
Game Over
Victory

Figure 14. Activity Diagram for Game Flow

7.4 User interface

We used the online Balsamiq tool to create the mockups for the UI.

7.4.1 Main Menu

Main Menu is the first window that appears when the application is opened, and includes 5 options: Play Game is the option to start the game, clicking on it directs the user to the "Choose Mode" window. Instructions informs the player about the game, gives a brief description about the different modes and how to play each.

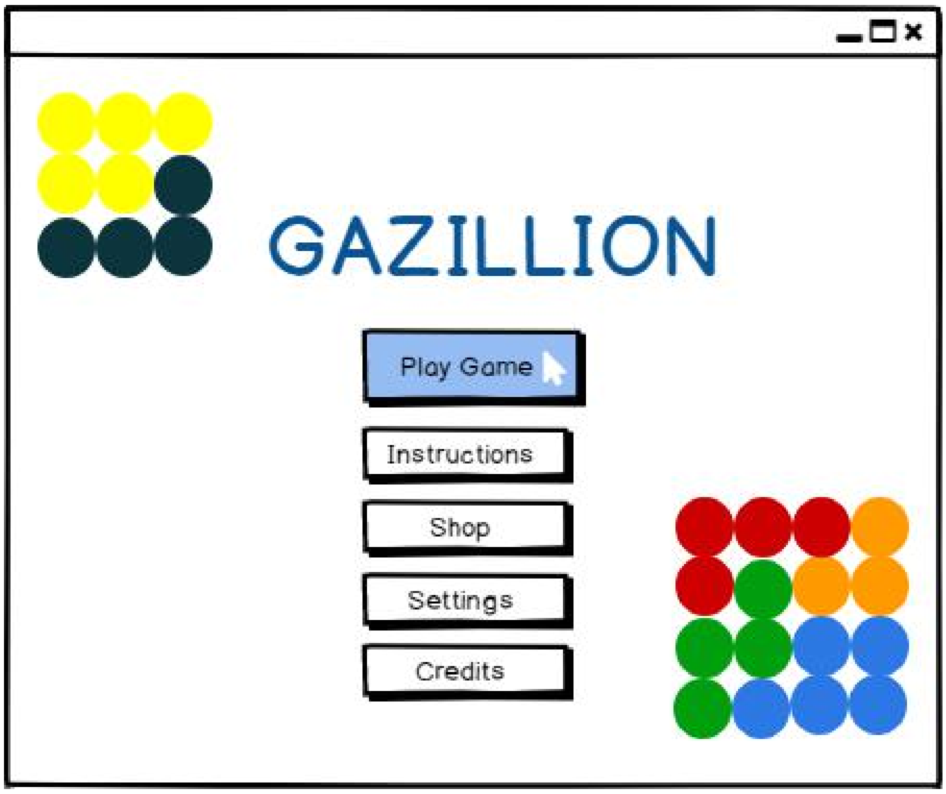


Figure 15. Main Menu

7.4.2 Modes Screen

Upon clicking the "Play Game" button in the Main Menu, the user is shown the different modes of the game to choose from. Player can go back to the Main Menu by clicking on the Back button.

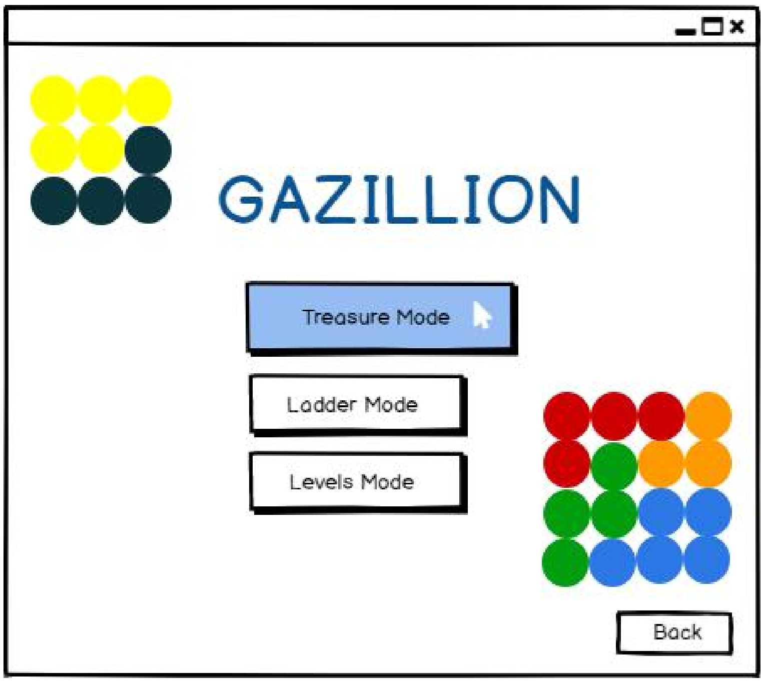


Figure 16. Modes Screen

7.4.3 Settings Screen

Clicking on the Settings button in the Main Menu opens the Settings window for themes and sound. Player can choose any of the available themes (themes he has purchased in the shop) by clicking on the Select Theme button below the preview. Player can additionally mute sound by clicking on the icon in the lower left or change the volume using the volume slider. Back button redirects Player to the Main Menu.

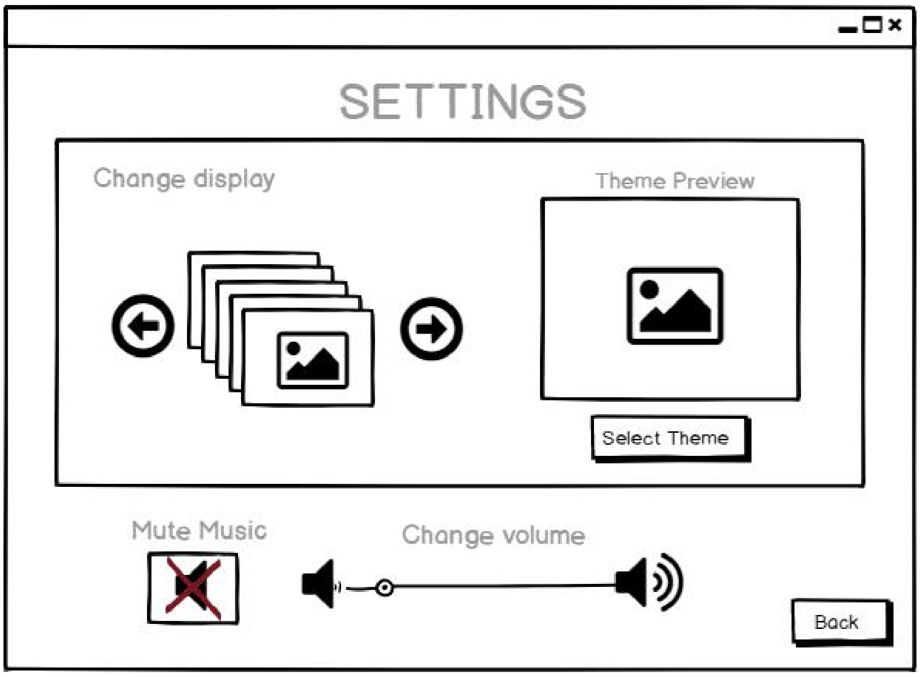


Figure 17. Settings Screen

7.4.4 Shop Screen

"Shop window" is opened by clicking on the Shop button in the Main Menu. Player can buy themes, treasure hints, health and time powers ups using the respective buttons. Player's budget (number of collected coins) is also displayed. Back button redirects Player to the Main Menu.

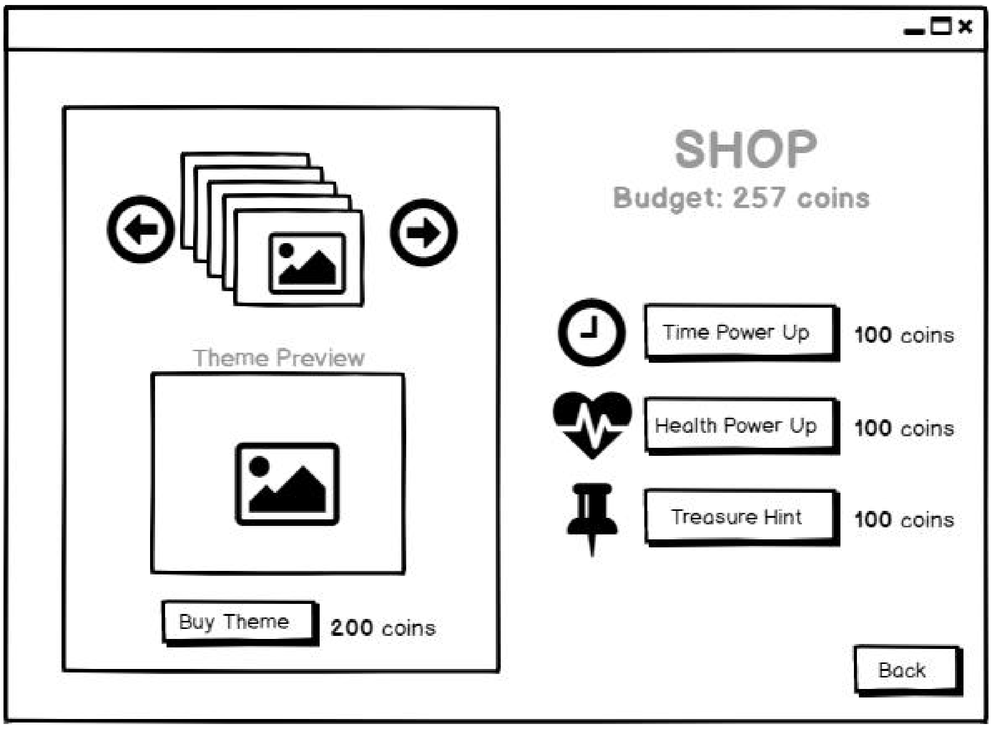


Figure 18. Shop Screen

7.4.5 Quadrillion Game screen

This is the gameplay screen of the Quadrillion game. On the top Player information is displayed: player name, health and coins. The Game board is on the left side of the window and on the right side the piece panel with all the game pieces is displayed. Player can select pieces from there and put them on the board after rotating or flipping them. That is also where the pieces are positioned when Player removes them from the board or when Player attempts to incorrectly place them.

On the bottom of the screen the remaining time is displayed, and the option to use time power ups (which increase the remaining time) and health power ups is right next to it. The number of power ups remaining is also displayed in parenthesis and if Player uses all of the power ups the button becomes disabled. "Reset" button removes all pieces from the board and places them in their initial slots. "Give Up" redirects User to the Window of the

Mode from where he started playing the game, after decreasing one unit from him health points.

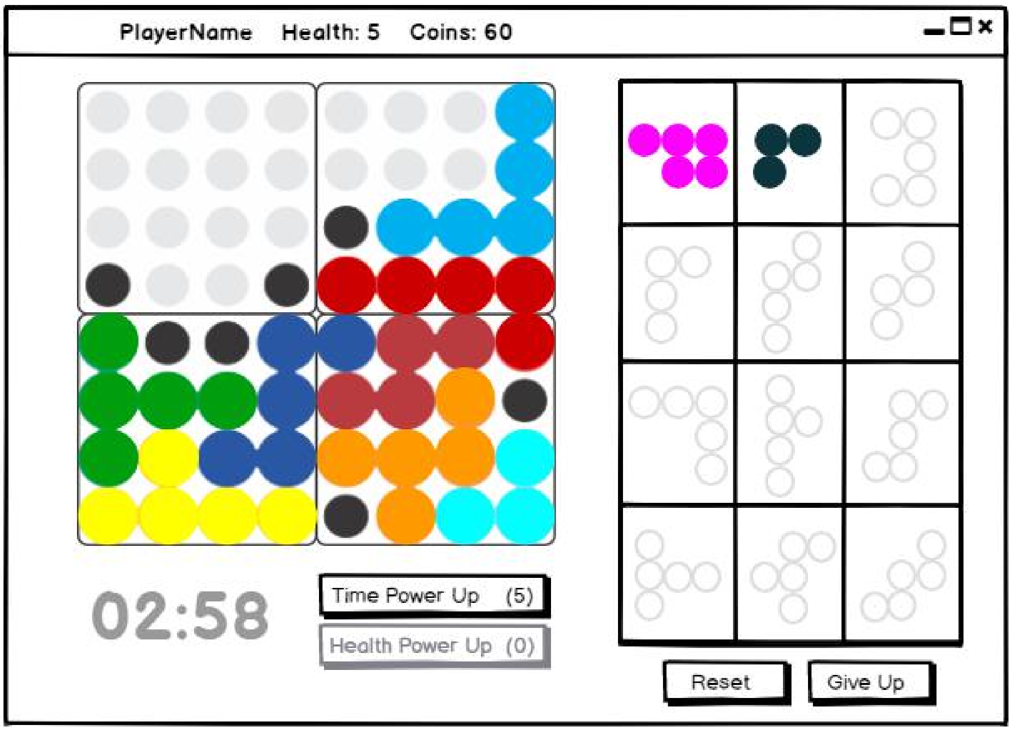


Figure 19. Quadrillion Game Screen

7.4.6 Level mode

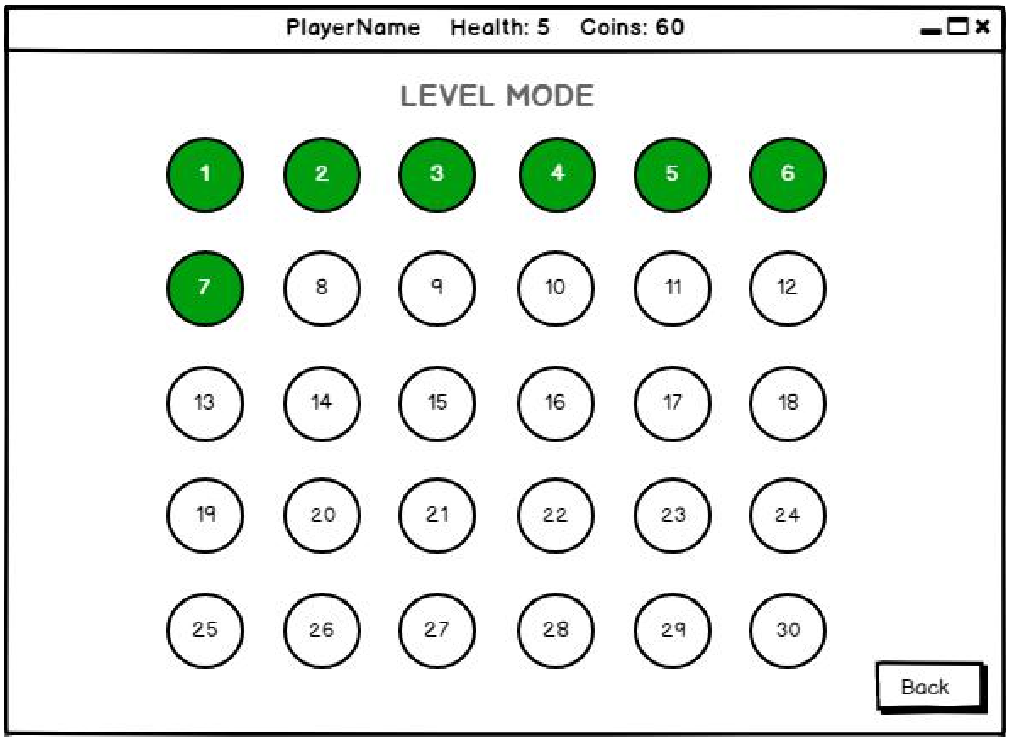
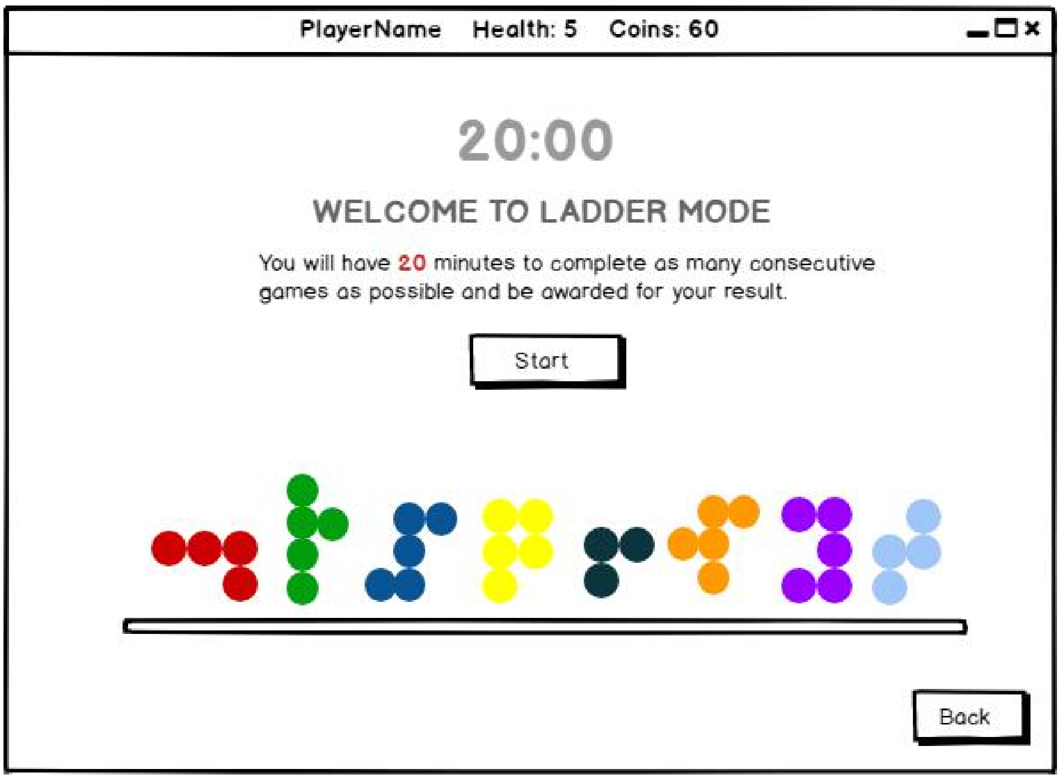


Figure 20. Level Mode Screen

This is the screen which appears when the user chooses the “Level Mode”. 30 levels in order of increasing difficulty are shown on the screen, and initially only the first one is unlocked. Player can only play the unlocked levels and he can unlock levels by completing the previous one. Clicking on a particular level generates a game with the appropriate difficulty level and redirects Player to the “Quadrillion Game” window. Back button redirects Player to “Choose Mode” Window.

7.4.7 Ladder Mode

This is the first screen which appears when Player enters Ladder mode. The total time is displayed at the top, and clicking on Start button starts the first random Quadrillion game. Back Button redirects Player to “Choose Mode” Window.


PlayerName Health: 5 Coins: 60

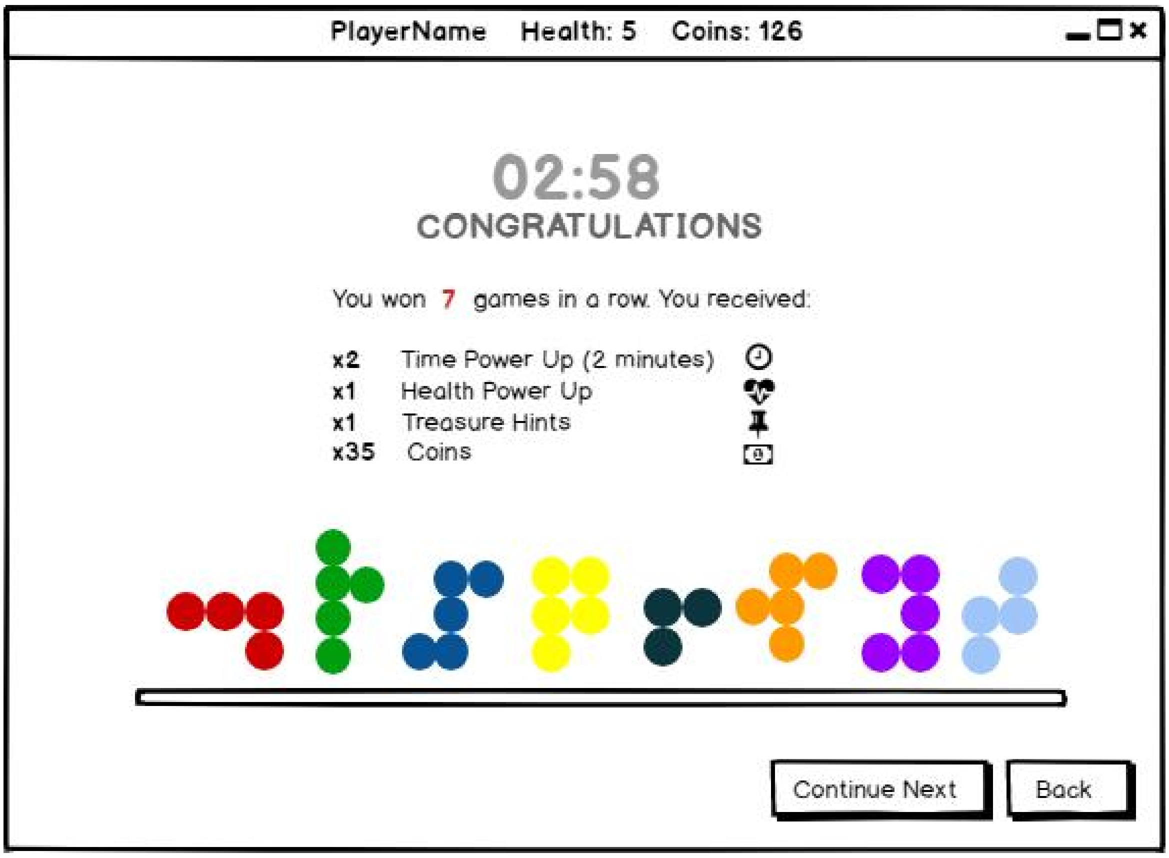
20:00

WELCOME TO LADDER MODE

You will have 20 minutes to complete as many consecutive games as possible and be awarded for your result.

Figure 21. Ladder Mode Initial Screen

This is the intermediate screen appearing between two random games. The remaining time is displayed at the top, and information about the number of games won so far and the award for winning the last game is also displayed in the screen. Clicking on Continue Next button starts the next random Quadrillion game. Back Button redirects Player to “Choose Mode” Window.



You won 7 games in a row. You received:

x2	Time Power Up (2 minutes)	
x1	Health Power Up	
x1	Treasure Hints	
x35	Coins	

Figure 22. Ladder Mode Intermission Screen

7.4.8 Treasure Mode

Upon choosing the Treasure Mode, this screen is displayed. On the top of the screen, the Player's information is displayed. On the left side of the window, the treasure map with 36 levels (in the form of buttons) is displayed. The unlocked levels are coloured in green, and Player can only choose among these levels to play and unlock the adjacent levels. Initially only the corner levels are available. Below the map, Player can see the number of pieces he has collected so far and can learn where the next treasure is hidden by clicking on the "Reveal Treasure". A level that contains an uncollected treasure is indicated by a red circle.

On the right side of the screen, there is a panel that contains the treasure pieces that the Player has collected so far. The "Play Final Game" button just below it is enabled only when the Player has collected all of the 12 puzzle pieces. The Final Game (the quadrillion game that decides the final outcome of the whole Gazillion game) is initiated after Player clicks on it.

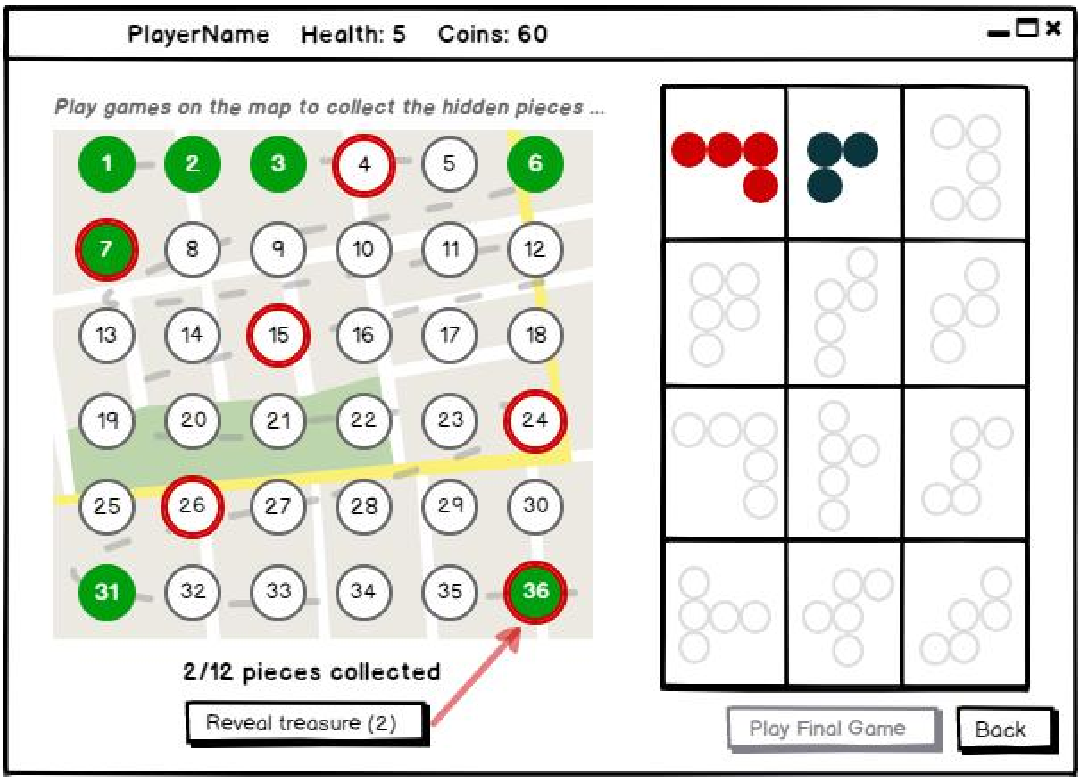


Figure 23. Treasure Mode Map View Screen

8. Conclusion

In this report, we have analysed the initial design of our game "Quadrillion". The report includes the requirement analysis and the system models.

The analysis consists of a summary of the functional and non-functional requirements of the project. Regarding the functional requirements, we have explained the behaviour and functionality provided to the player. The non- functional requirement contains performance-wise features of the application like accessibility, usability, portability, gameplay stability etc.

Moreover, the report depicts the overall system model by making use of diagrams such as use case diagrams, sequence diagrams, class, activity and state diagrams. Each diagram illustrates a specific aspect of the project functionality. Hence, this report serves as a foundation for the latter implementation and design of the project.

9. References

[1] “Discover Our SmartGames Collection.” *IQ Puzzler Pro - SmartGames,* https://www.smartgames.eu/uk/our-smart-story. [Accessed: Mar 10, 2019].

[2] “Quadrillion.” www.smartgames.eu/uk/one-player-games/quadrillion. [Accessed: Mar 05, 2019].

[3] “QUADRILLION - keep on playing forever”. www.smartgamesandpuzzles.com/inventor/Quadrillion.html. [Accessed: Mar 05, 2019].