Bilkent University

Department of Computer Engineering

CS319 Term Project

QuadZillion

# Iteration - I

# Analysis Report

**GROUP 1-G**

Melike Arslan

Enver Yiğitler

Ufuk Bombar

Kasım Sarp Ataş

Berk Güler

Supervisor: Eray Tüzün

Table of Contents

1 Introduction

2 Overview

3.1 Functional Requirements

3.1.1 Play Game

3.1.2 Level Menu

3.1.3 Create Custom Puzzle

3.1.4 Settings

3.1.5 Credits

4 Nonfunctional Requirements

4.1 Implementation Requirements

5 System Models

5.1 Use Case Model

5.1.1 Use Case Analysis of Play Game

5.1.2 Use Case Analysis of Pause

5.1.3 Use Case Analysis of Play

5.1.4 Use Case Analysis of Settings

5.1.5 Use Case Analysis of Credits

5.2 Dynamic Models

5.2.1 Start Game

5.2.2 How to Play

5.2.3 Settings

5.2.4 Player piece interaction

5.2.5 Clear Board

5.2.6 Pause

5.3 Object and Class Model

5.3.1 Analysis of the core game

6. References

1 Introduction

It’s a Sunday morning. The weather outside is rainy and foggy, you don’t feel like going outside but you have nothing to do, why not play “quadZILLION”?

2 Overview

For our CS319 Term Project we were assigned to design and implement the digital version of the board game “Quadrillion”. It is an adventurous game challenging your brain with various puzzles. It is for every age and it guarantees hours of enjoyment. Our version of the game is called “QuadZillion”. QuadZillion is a desktop game that allows a player to complete various logical puzzles. The main objects of the game are 4 double-sided black and white grids and 12 colorful puzzle pieces. These objects are the main focus of the game.

Gameplay

The player will have to complete each level in order to reach the next level. Each level is made up of 4 double sided grids with 16 circles on them each. These circles are to be the spaces for the puzzle pieces which will be placed on the tiles. The player is expected to put the pieces on the board without any spaces left other than the given ones as a result. All pieces have to be used for each level. There will be a time counter for each level and this time counter will act as a score for each level. Unless the player finishes the level, he/she can’t get to the next level. Once the player completes the level, the next level will unlock and so on. The levels get more and more harder as the player proceeds.

3 Functional Requirements

3.1 Play Game

User can start the game by clicking the play game button. This button opens the level menu screen.

3.2 Level Menu

Level menu shows the preset levels. Player unlocks a level by solving a previous level. If player is stuck at a level than player will have a chance to get hints from the game. The amount of hints are limited and hints can be gained

by solving puzzles. Player is able to play any unlocked level but only solving the last level unlocks a locked level.

3.2 How to Play

There is a tutorial section for new players in the main menu. This tutorial explains game’s rule, game’s mechanics and the level system.

3.3 Create Custom Puzzle

Players are able to create their own puzzle and play it. Game saves the created level and player can play them any time.

3.4 Settings

Player can adjust the SoundFX, MusicFX through the settings menu which can be accessed through the main menu. Player can also change the language and the theme of the game to further customize the game to their desires.

3.5 Credits

Player can learn about the developers of the game and can access the GitHub page of the game through the credits option. They can either make contribution to the game or request new features from the developers through GitHub.

4 Nonfunctional Requirement

The target audience for QuadZillion is those interested in puzzle and brain teaser kind of games. There is no prerequisite to play this game, anyone can learn the rules in a short amount of time and start playing. The rules and mechanics are relatively simple, however the game itself can be quite challenging as the difficulty level increases. How to play section will be provided for the players to get them into the game as quickly as possible and start solving the puzzles.

4.1 Implementation Requirements

Game will be developed in Java language as a desktop application, thanks to JVM this game can be played on any operating system. The only requirement is to have Java Virtual Machine installed on the target computer. The standard GUI library of Java which is JavaFX, will be used in our project to make visually appealing interfaces, which is really important as users in this age can be turned off quickly if the visuals are not up to the standards of the current technology. As for the reliability, players can exit the game

5. System Models

5.1 Use Case Model

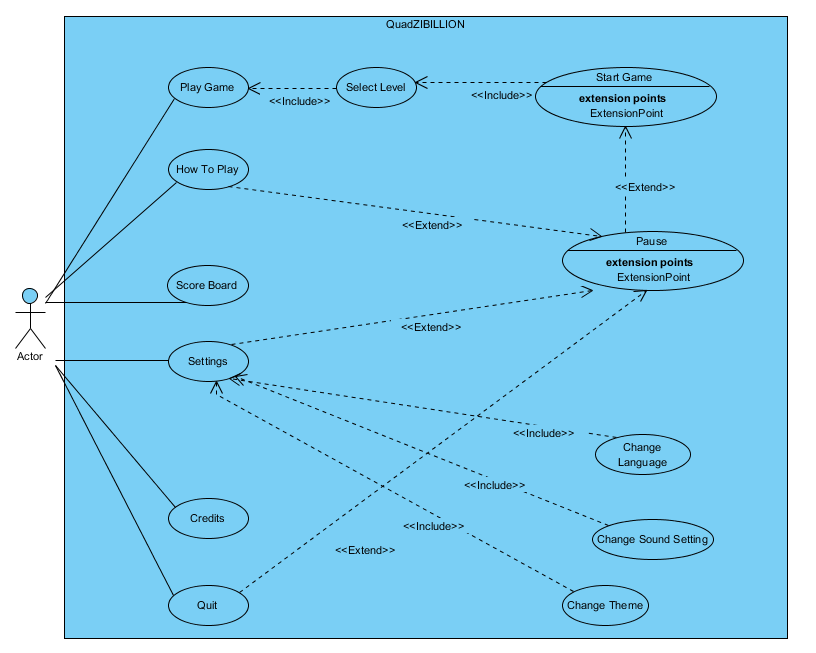


Figure 5.1.1 *Use Case Diagram of QuadZillion*

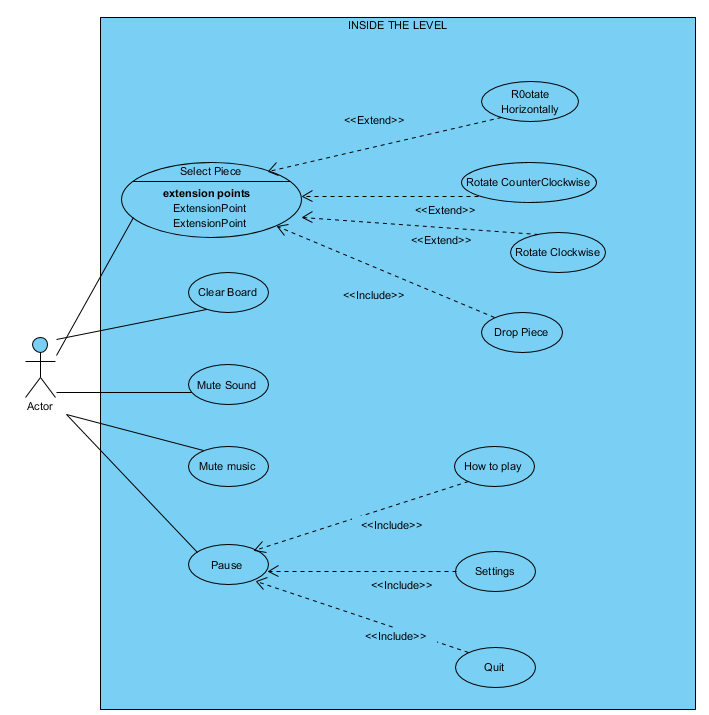


Figure 5.1.2 Inside Level Diagram *of QuadZillion*

5.1.1 Use Case Analysis of Play Game

**Use Case: Play Game**

Actor: Player

Pre-Conditions: Player must be in the main menu in order to select “Play Game”.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “Play Game” button in the main menu.

Exit-Condition: In order to exit, player must select level and then when game starts user press pause button and choose to quit. Or presses “ESC” to return main menu.

Successful Scenario:

1. Player selects “Play Game”.

2. Player selects “Selects Level” and then choose the desired level.

3. Player starts the game.

4. Player solves the given puzzle.

5. Player is given an interface to go to next level, repeat or return to the main menu.

6. Player chooses the desired next option.

Different Scenarios:

1. Player quits the game before solving the puzzle given and go to the main menu or rage quit the game entirely since the puzzle is too hard to solve.

2. Player returns to the main menu without choosing a level.

5.1.2 Use Case Analysis of Pause

**Use Case: Pause**

Actor: Player

Pre-Conditions: Game must be started in order to pause the game.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “Pause” button in the gameplay screen.

Exit-Condition: In order to exit, player must select the “Quit” button and return to the main menu or press “ESC” on the keyboard to close the pause screen (continues the game since player is determined to solve the hard puzzle)

Successful Scenario:

1. Player presses the pause button in the screen.

2. Player chooses among “Settings”, “How to Play” or “Quit” buttons and clicks it.

3. The corresponding action will be executed and Player will have provided with the interface desired.

Different Scenarios:

1. Player changes mind and decided to continue to play the game.

2. Presses “ESC” to close the pause menu.

5.1.3 Use Case Analysis of Play

**Use Case: How to Play**

Actor: Player

Pre-Conditions: Player must be in main menu in order to select How to Play.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “How to Play” button in the main menu.

Exit-Conditions: Player presses “ESC” to return to main menu.

Successful Scenario:

1. Player presses “How to Play” button in the main menu.
2. Player understand the game rules.
3. Player presses the “ESC” to return to main menu.

Actor: Player

Pre-Conditions: Player must be in main menu in order to select How to Play.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “How to Play” button in the main menu.

Exit-Conditions: Player presses “ESC” to return to main menu.

Successful Scenario:

1. Player presses “How to Play” button in the main menu.

2. Player understand the game rules.

3. Player presses the “ESC” to return to main menu.

5.1.4 Use Case Analysis of Settings

**Use Case: Settings**

Actor: Player

Pre-Conditions: Player must be in main menu in order to select Settings.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “Settings” button in the main menu.

Exit-Conditions: Player presses “ESC” to return main menu.

Successful Scenario:

1. Player presses “Settings” button in the main menu.
2. Player given the settings interface (which contains Change Language, Change Sound Settings, Change Theme)
3. Player chooses one of the options.
4. Player adjust desired settings.
5. Player presses “ESC” to return main menu.

5.1.5 Use Case Analysis of Credits

**Use Case: Credits**

Actor: Player

Pre-Conditions: Player must be in the main menu in order to select “Credits”.

Post- Conditions: There is no particular post condition.

Entry-Conditions: Player must select “Credits” button in the main menu.

Exit-Conditions: Player presses “ESC” to return to main menu.

Successful Scenario:

1. Player presses the “Credits” button.
2. Returns to the main menu.

5.2 Dynamic Model

5.2.1 Start Game

Scenario: Player starts the game

Player enters to the game and is directed to the main menu. From the main menu player clicks the “Play Game” button and in order to play one of the premade levels, user clicks the “Select Level” option. Level Manager returns the available levels and Game Manager loads the game.

5.2.2 How to Play

Scenario: Player selects How to Play from the main menu

After starting the game, player might not want to jump straight in to the game and might want to learn the rules and the game mechanics so player clicks “How to Play” button. Here the rules and basically every information needed to play the game is shown.

5.2.3 Settings

Scenario: Player selects Settings from the main menu

After starting the game player might want to change the settings and customize it according to their desires, so player clicks “Settings” button. Here the player is given the option to change the language of the game, the theme of the game and lastly change the sound levels. Player can return back to the main menu after changing certain settings or without making any changes.

5.2.4 Player Piece Interaction

Scenario: Player interacting with pieces

Player wants to insert pieces in to the board so clicks one of the pieces. After clicking a piece, the player can either rotate the piece horizontally using the SPACE key on the keyboard, rotate it counterclockwise, rotate it clockwise using the A and D keys on the keyboard and drop the piece on the board (drag &drop) or somewhere else in the screen using the mouse.

5.2.5 Clear Board

Scenario: Player selecting the Clear Board option

While inside the game player chooses the Clear Board option to reset the game to its starting position. If there was a given piece in the beginning of the puzzle, it remains after resetting the game. Player might want to choose this option when they’re stuck with the puzzle and they might want to take another clear look at the puzzle. It could also be due to a misclick, for instance putting the piece on the wrong place.

5.2.6 Pause

Scenario: Player wants to pause the game

While inside the game, player can pause the game anytime they want in order to take a break. Player will be shown the options to adjust the settings, quit the game completely or go to the how to play section which is also accessible from the main menu.

5.3 Object and Class Model

5.3.1 Analysis of the Core Game

Before making the implementation version of objects and object models we decided to analyze the game and make an abstract diagram that represents the the game. We have decided to move forward with that decision because, going from basic to high level is a well thought design approach.

The problem we faced was the representation of the board and the pieces in an efficient way. We choose to assemble the abstract objects such as the pieces and the grids an object called Table. Table is responsible for storing and manipulating the data on other objects. It contains 3 different types of classes which are Piece, Grid and TableData.

Piece, is an abstract class that has certain attributes to represent a piece. First it has the position on the world which is stored to the object, Position. Position contains the x and y coordinates of the start of the *hitbox* of the piece. In other words, Position object consists of two integers to represent the discreet position on the table. The hitbox, is represented by the PieceData object which inherits from the abstract class Data. It contains the present representation of the piece in a 2 dimensional array. The operations of flipping and rotating is performed by this class. PieceData inherits from the Data class and has special functions for the pieces. Piece has 12 child classes that represents each piece in the game. Every piece’s PieceData object is initialized by the constructor. Overall, Piece class holds and represents a piece.

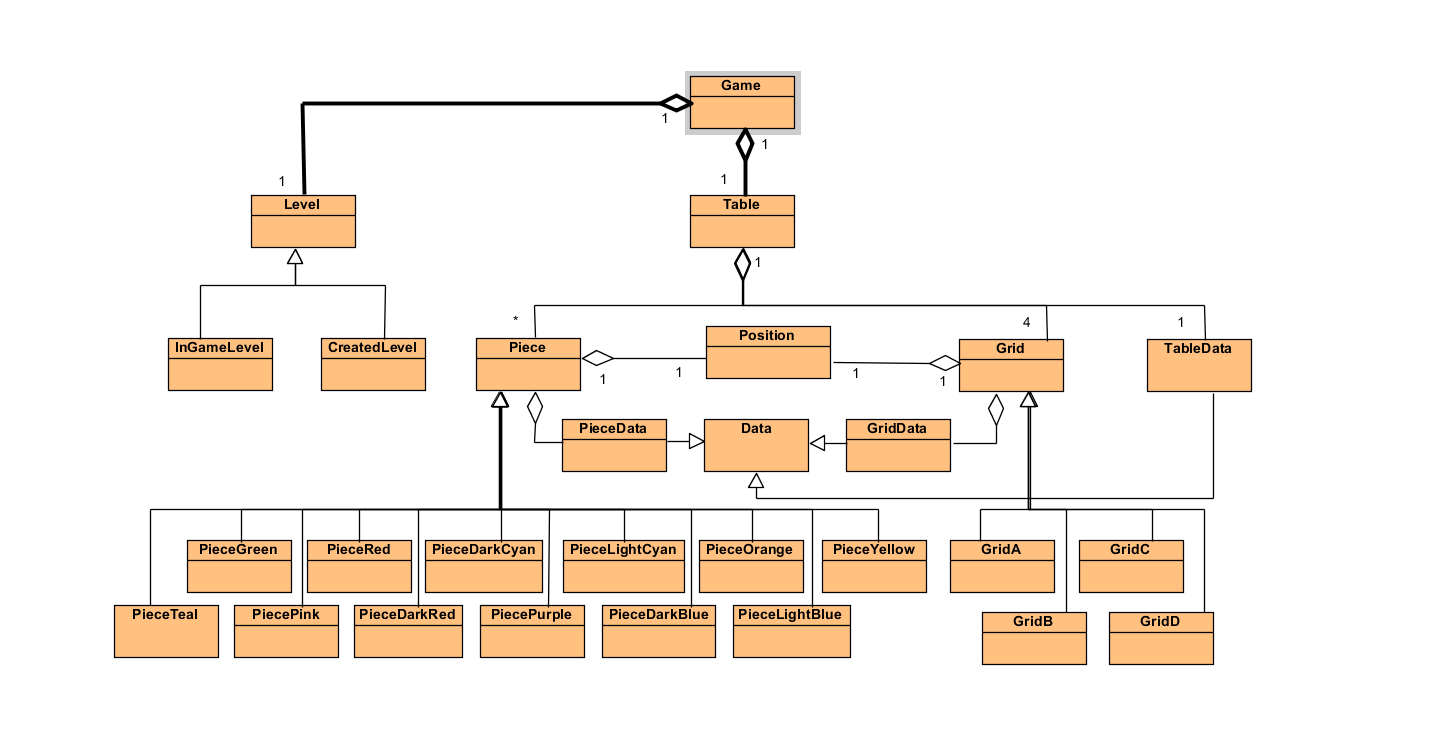


Figure 5.3.1 *Core Model of QuadZillion*

The Grids are represented with the same way like the pieces. The Grid class is an abstract class which has the following attributes; Position and GridData. Position as explained before represents the position but in this case it represents the position of the grid relative to the table. GridData is used for holding the data of a grid. The grid in the quadrillion is 4 by 4 therefore the size of the GridData is fixed to 4 by 4. GridData is also inherits from Data class.

Lastly, the system of pieces and grids are stored in the class Table. Table has 4 Grid objects since in the original game, it is played with 4 grids. When the table is created the grids are passed to the constructor with adequate positions. In that point we don’t want any illegal positions such that two or more grids are intersecting. When the system is created the TableData object is created and overwritten. After all, the data is stored on the TableData object located inside Table class.

Level is located in the Game class alongside with Table. When a new level is loaded there are two possibilities. First one, the Level can be a default level which means it is a level located in the original game’s booklet. Since we know the solutions we can give hints to the user this is why InGameLevel inherits from the level. The second possibility is that the level to be a created level. This means the level contains additional information of how the grids are arranged. This is why CreatedLevel inherits from the Level class.

6 References

1) https://nerdarchy.com/board-games-review-quadrillion-logic-game/

2) https://www.smartgames.eu/uk/one-player-games/quadrillion