QuadZillion

CS319 Iteration II Analysis Report

Bilkent University Department of Computer Engineering

Group 1G Supervisor: Eray Tüzün

Berk Güler
Enver Yiğitler
Kasım Sarp Ataş
Melike Arslan
Ufuk Bombar

Table of Content

1. Introduction

Board games are daily games that people play on their free time when they get together with their friends or by themselves. Board games usually include a main board and characters or pieces to put on the board according to the game.

Quadrillion is a board game that challenges the player to use logical thinking trying to solve multiple puzzles using Tetris-like pieces. These challenges push the player to develop their cognitive abilities. It also encourages critical thinking and requires players to come up with creative ways to solve the given puzzles. The puzzles consist of 4 magnetic double-sided grids and 12 colorful pieces. In each puzzle the placement of the grids is different creating various levels for the game. On each grid there are forbidden points, the pieces cannot be placed on these points. When the user finishes the puzzle he can try another level. The player can also create his/her own level. The pieces will fit no matter how the grids are placed creating an infinite amount of solutions and levels.

Our CS319 Term Project is a digital version of Quadrillion. We named this version “QuadZillion”. It is a 2-D puzzle game. The motivation behind developing a digital version of a game that already exists in the form of board game is to have an alternative with an appealing and user-friendly interface available on computers. Since the board game version is 3-D we made pieces in a way that they can rotate and flip. The players can create their custom levels like the board version. In addition to the board game, our version will include a timer for each level and player will be able to save the game to be able to continue later. Further, players can challenge their friends. They can compare their achievements using the scoreboard feature of QuadZillion.

2. System Proposal

2.1 Functional Requirements

2.1.1 Initial Start of the Game

Upon starting the game, user is greeted with a main menu that consists of five buttons, namely Play Game, How to Play, Credits, Settings and Quit. There will be a relaxing background music that the user can mute.

2.1.2 Play Game

User will be directed to game mode selection menu consequent upon pressing the Play Game button. Here the user can choose from the following game modes: Vanilla Mode, Extended Mode and Puzzle Mode.

2.1.3 Vanilla Mode

As the name suggests, the gameplay will be identical to the original board game. User will be able to move pieces and place them on the board. There will be set amount of premade levels and the user will be able to choose from them. Levels will be taken from the original game. User will be able to experience the game as it was intended by the original makers.

2.1.4 Extended Mode

In this mode we will extend the board and the pieces in order to increase the variety of pieces and amount of possible games. The pieces and boards will be randomly generated, the algorithm will ensure that every generated game will have at least one solution. See Figure 1 as an example.

2.1.5 Puzzle Mode

This mode allows the user to experience Quadrillion in a different way. While the game play is similar to original game in a way that there are pieces and board to interact with, main goal of this mode is to complete a puzzle picture. The image that the user is trying to get to will be divided to pieces with different shapes. The user will try to place the pieces on the board so that they can reach to desired image.

2.1.6 How to Play

This scene is available from the main menu. Upon entering the scene, the user will have access to a video which explains the rules of the game, mouse/keyboard controls and the different game modes.

User will get familiar with the game quickly and wouldn't have to waste unnecessary time trying to learn everything on their own with trial and error method.

2.1.7 Settings

After pressing the Settings button, the user will be given the option the alter several different settings of the game, such as changing the them, adjusting the music level or completely muting the music.

2.2 Nonfunctional Requirements

This section includes the nonfunctional requirements. We evaluate them into the following titles.

2.2.1 Usability

The application will be distributed with an executable jar file. This executable can be run on the target machines as long as the target machine's operating system supports JRE (Java Runtime Environment) and it is installed. User will be able to start the application by either double clicking the jar file, running "java – jar fileName.jar" command through the

terminal or by compiling the source code themselves in their computer. The application does not require a specific operating system or specific hardware requirements it can be run on many different machines. This ease of access and having multiple options to play the game allows us to appeal to a wider audience.

We also have a separate scene designated to teaching the user how to play and how to interact with the rest of the application. This scene contains information about the game and play experience. It is accessible through the How to Play button located in the main menu. By utilizing this scene user can get familiar with the rules of the game and learn how to play it.

2.2.2 Reliability

In case of an unexpected crash, the user's progress will be saved, and the user will be able to access it after relaunching the application. Since the application does not require any internet connection and every component of the application is distributed through a single jar file (includes the video and assets), user won't face any unexpected results in the case of lost internet connection.

2.2.3 Performance

Since the screen refreshes 60 times a second every action is seemingly instant and there is no waiting time in between scene changes and piece movements. The application is not resource heavy and doesn't require modern hardware.

2.2.4 Supportability

The application supports both x86 and x64 architectures. It will be adaptable for additional features to be added in the future. For instance, it is possible to add new levels and new game modes to the application. The game will also be able to change according to new technologies in the future.

2.2.5 Implementation

The programming language used for this game is Java for desktop usage. In the implementation of the system GUI part will make use of the now external JavaFX library (it was a built-in until Java 11). JavaFX allows the use of MVC design pattern and also supports CSS (Cascading Style Sheets) to make visually appealing graphical components. Java supports the Object-Oriented Programming paradigm and thus was chosen since it was the best fitting programming language for the application's requirements.

2.2.6 Packaging

The application will be distributed through a .jar file, which is basically an executable zip-file that contains everything necessary to run the application, namely the source code, dependencies, external libraries and assets. There are no extra required dependencies to be installed (except JRE) which makes it convenient to distribute the system.

2.2.7 Legal

There is no legal requirement for QuadZillion. It does not contain any copyrighted materials. The game is released under MIT License, see: www.opensource.org/licenses/MIT for further information.

3. System Models

3.1 Use Case Model

The following diagrams best demonstrates the use case of the system. Figure 3.1.1 denoted for the interactions with user and main menu. Figure 3.1.2 denotes the basic interactions of inside level.

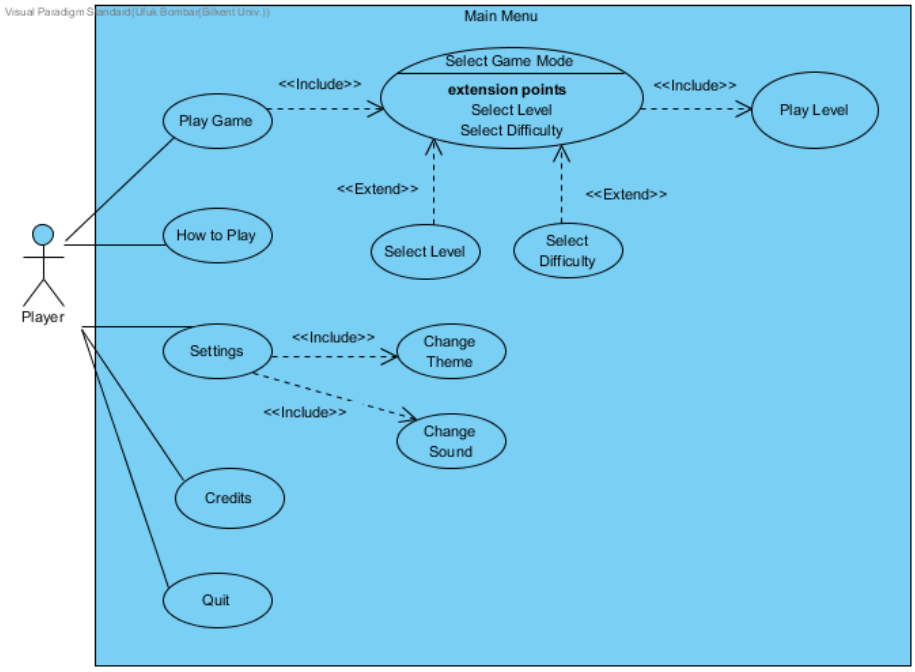

Figure 1 Use Case of Main Menu

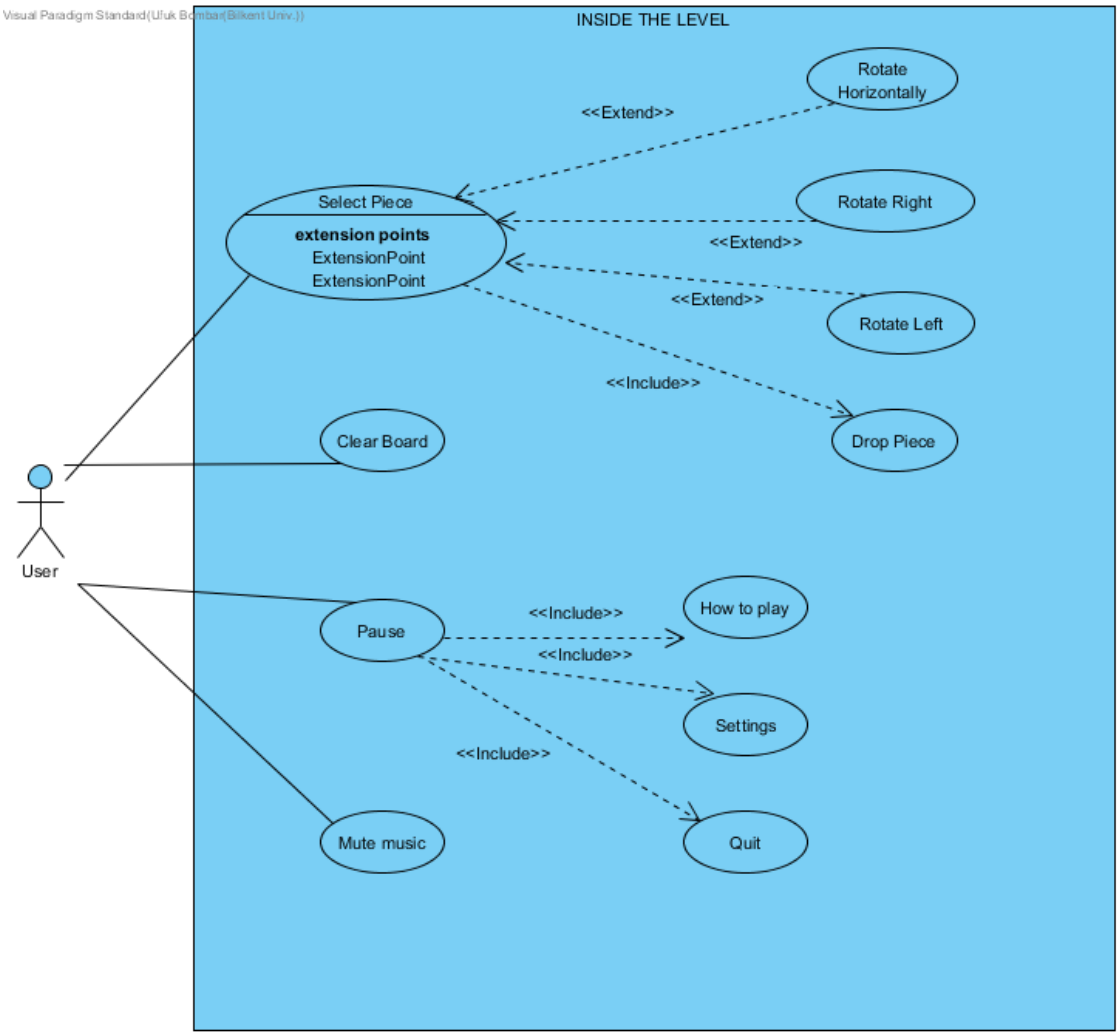

Figure 3.1.1 Use Case of Inside Game

3.1.1 Scenarios

3.1.1.1 Use Case Analysis of Play Game

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-Conditions	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.

3.1.1.2 Use Case Analysis of 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.

3.1.1.3 Use Case Analysis of 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.

3.1.1.4 Use Case Analysis of 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.

3.2 Dynamic Model

3.2.1 Sequence Diagram

3.2.1.1 Move Piece in Vanilla Game

This diagram shows the flow of action that is executed in case of a piece move. When mouse is released a method is executed inside the piece. This method invokes move method inside move checker object. In there the validity of move is calculated using tile matrix and main board. The status of move is returned as a type.

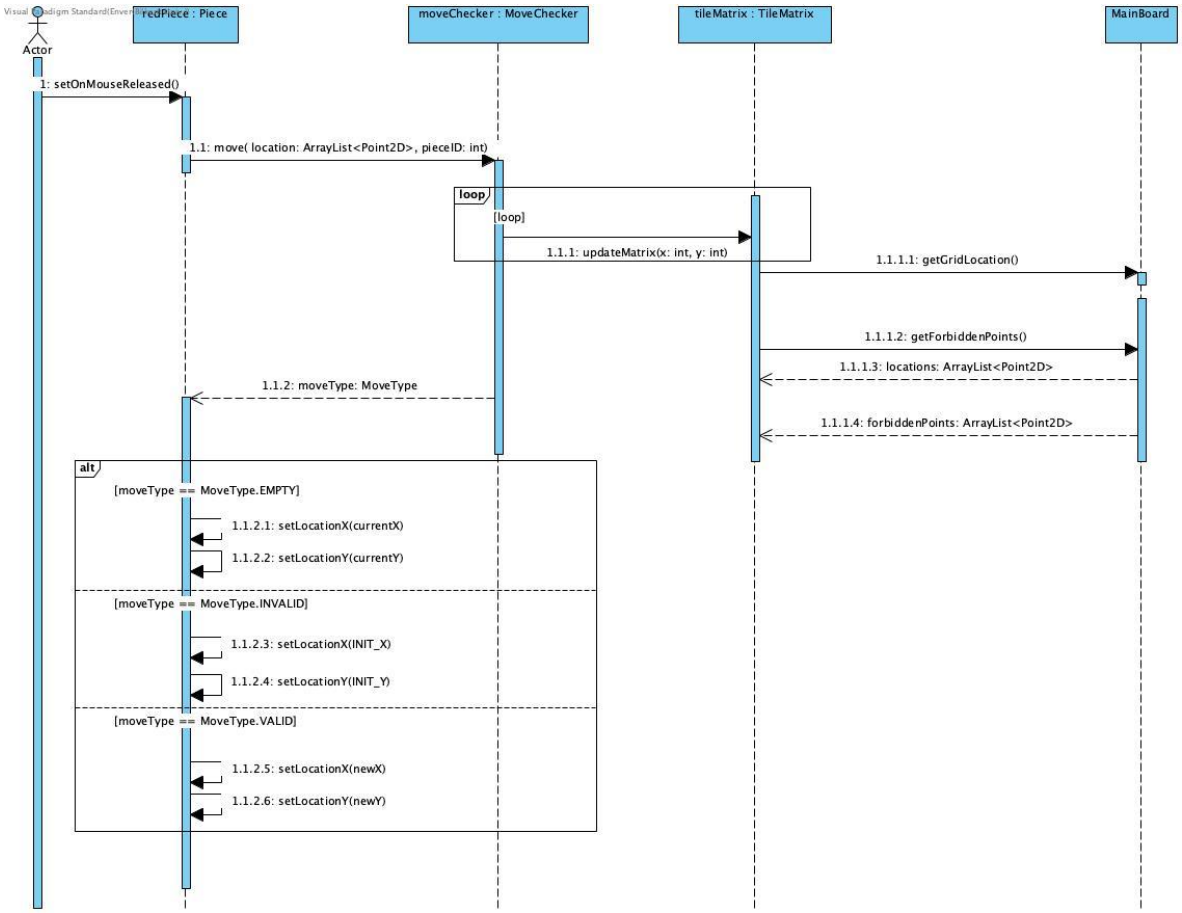

Figure 2 Description of how piece is moved

3.2.1.2 Create Extended Game

In this sequence, player will start executing the sequence by clicking the button, after that button click, the scene will be change to select difficulty screen the scene to request a difficulty to play. Scene change procedure is handled by Util. When user inputs the difficulty a new game pane is created. This creation will be handled by Pane, Level and Piece Generators. When the creation is finished a procedural generated board with generated pieces according to board will be presented to player.

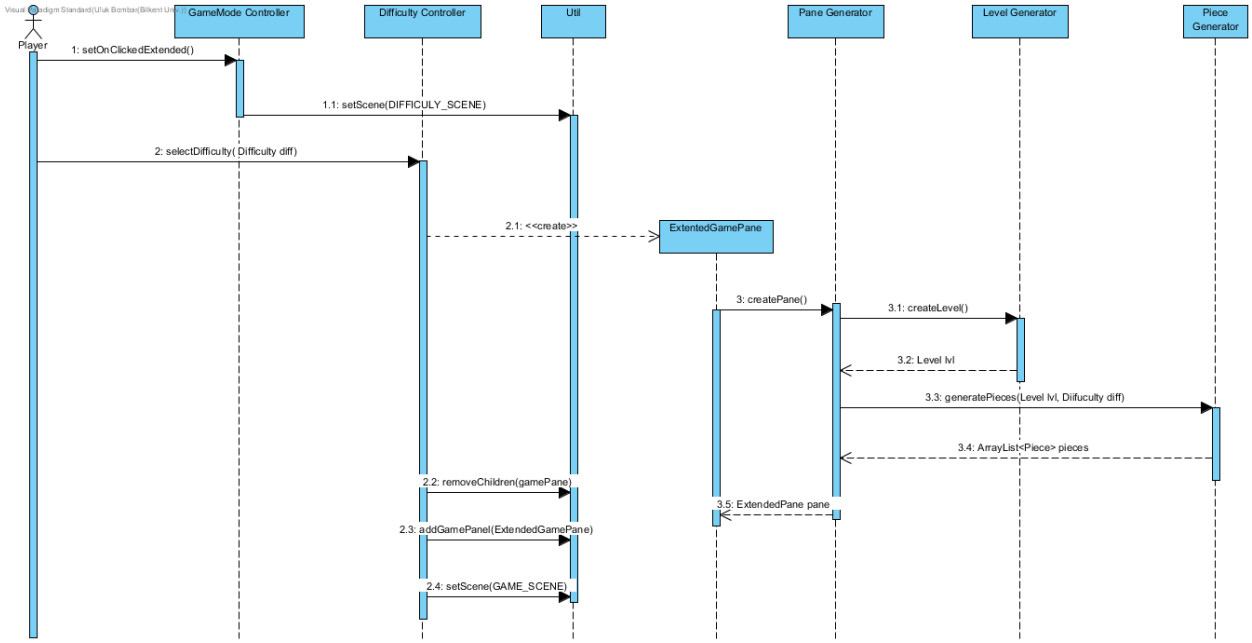

Figure 3 Description of how extended game will be created

3.2.1.3 Rotate Pieces

Piece will be turned clockwise by user's action.

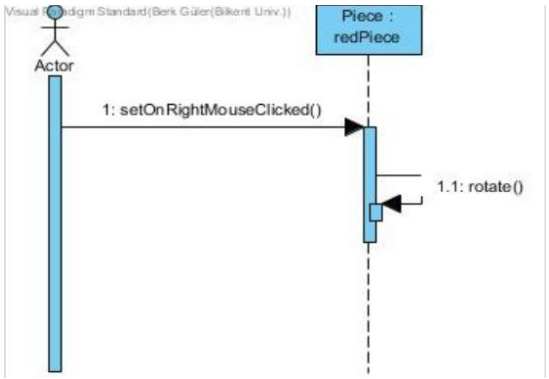

Figure 4 Description of how piece will be turned

3.2.1.4 Mirror Piece

Piece will be mirrored over its origin point by user's action.

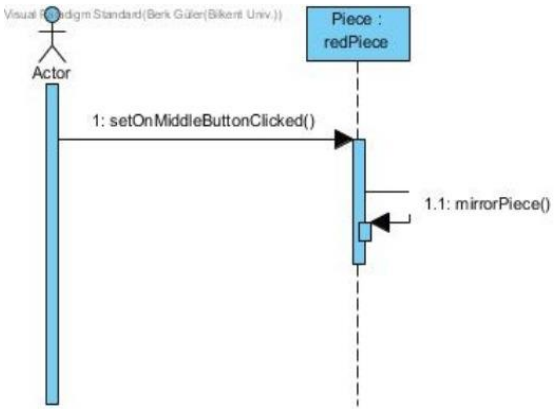

Figure 5 Description of how piece will be mirrored

3.2.1.5 Change Theme

Theme change operation is invoked by users input. For changing theme program gets the current scene via current game application. Game application gets the root map which is a set of roots. When a valid root has found a file loader loads the theme and updates the scene object.

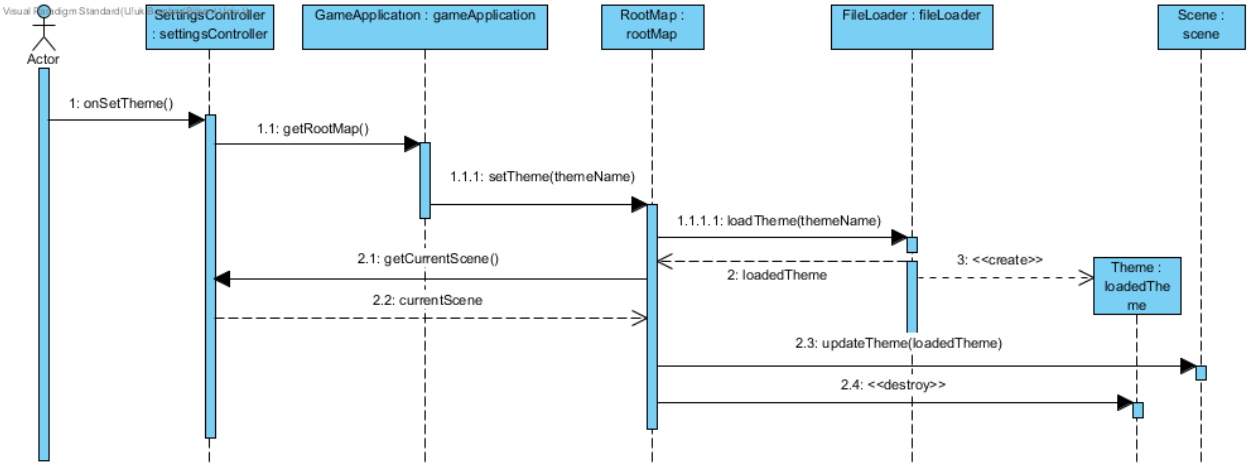

Figure 6 Description of how theme has changed

3.2.2 Activity Diagram

These diagrams describe the flow of events in a general sense.

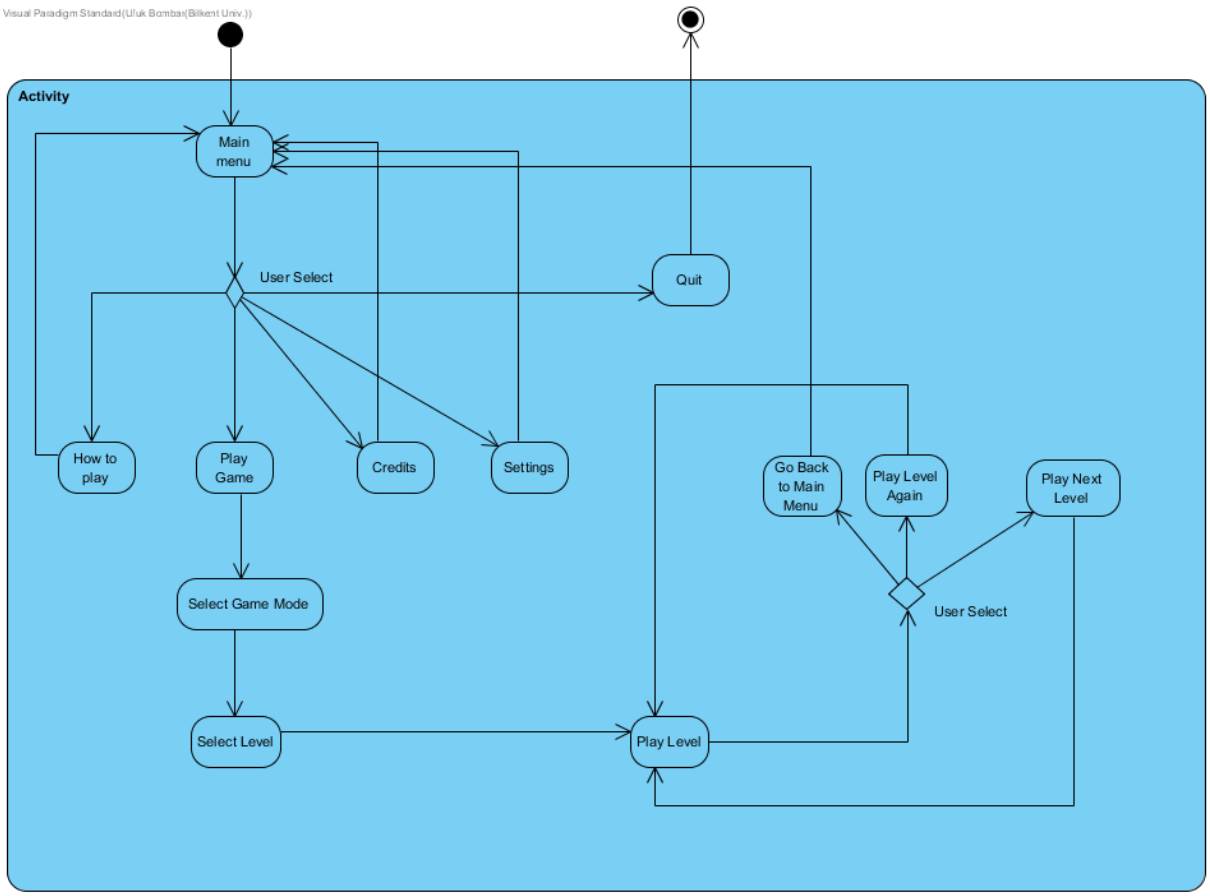

Figure 7 Activity diagram represents main flow

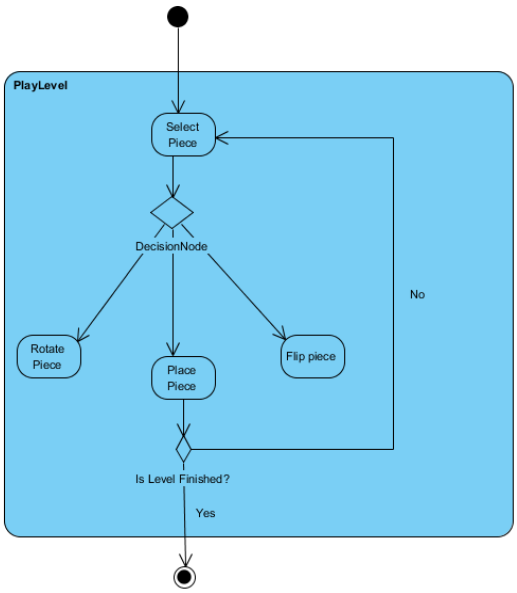

Figure 8 Activity diagram represents flow of inside level

3.2.3 State Diagram

This diagram shows the states of each piece.

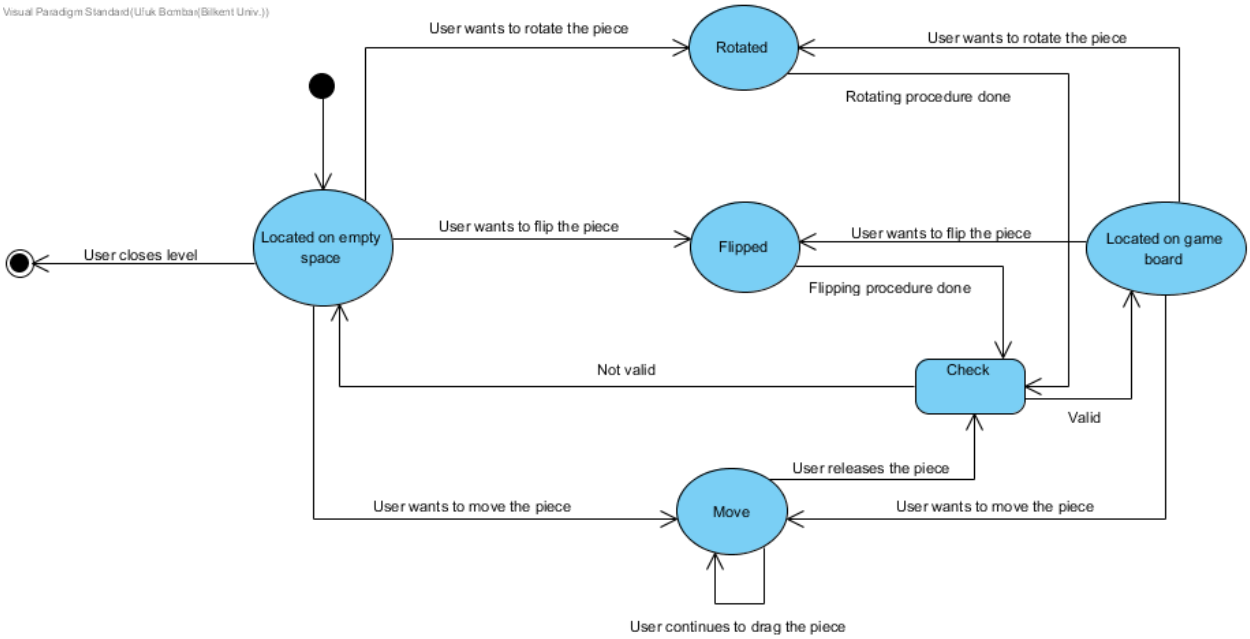

Figure 9 State Diagram that describes states of a piece

3.3 Object Class Model

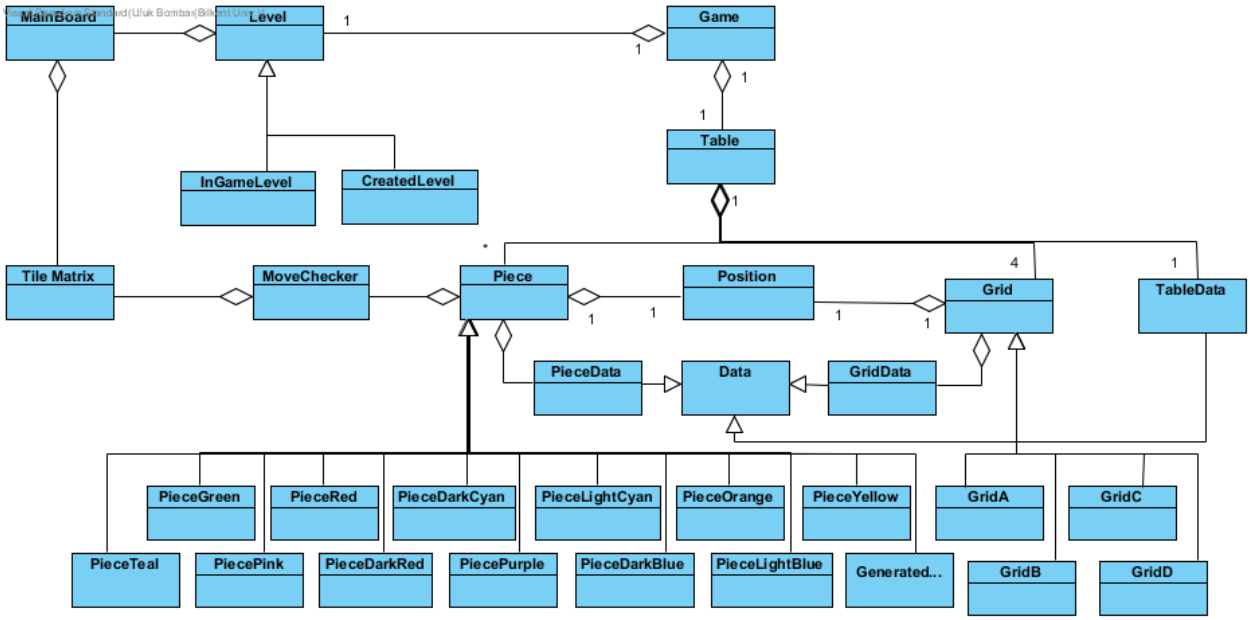

Figure 10 Class Diagram of core game

3.4 Mockups

Figure 10 Game Mode Selection Menu

Figure 11 Level Selection Menu for Vanilla Mode

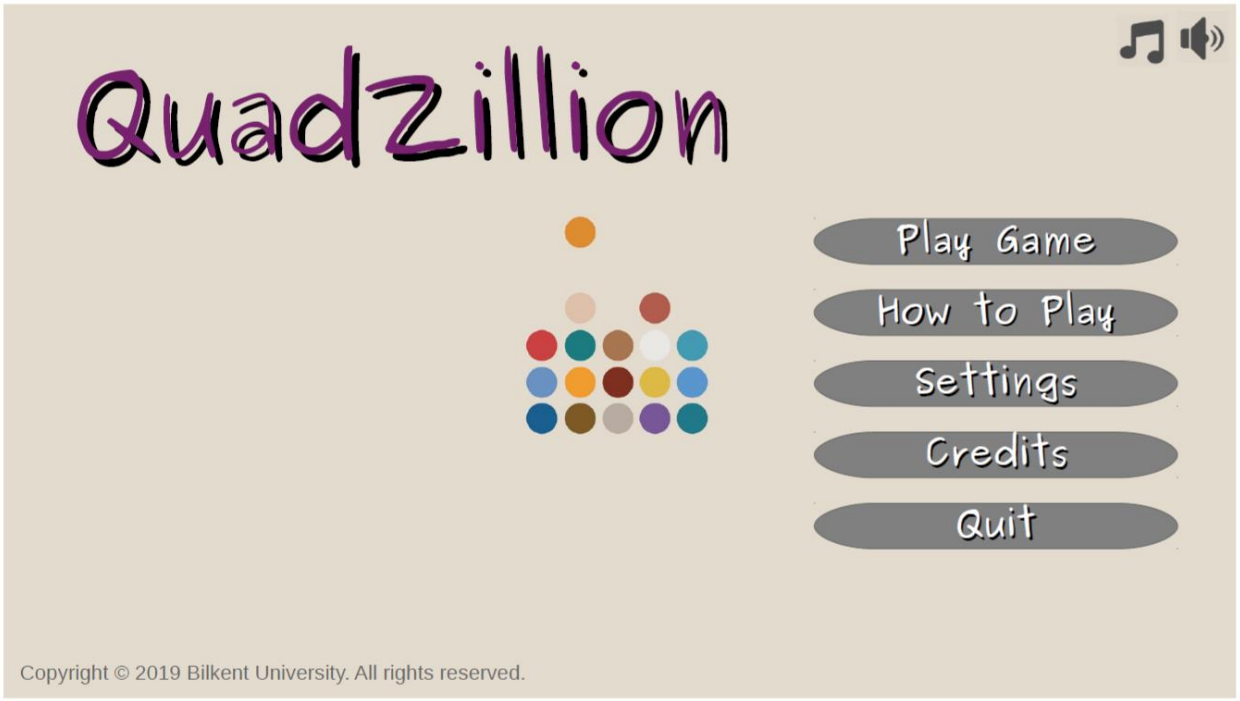

Figure 12 Main Menu

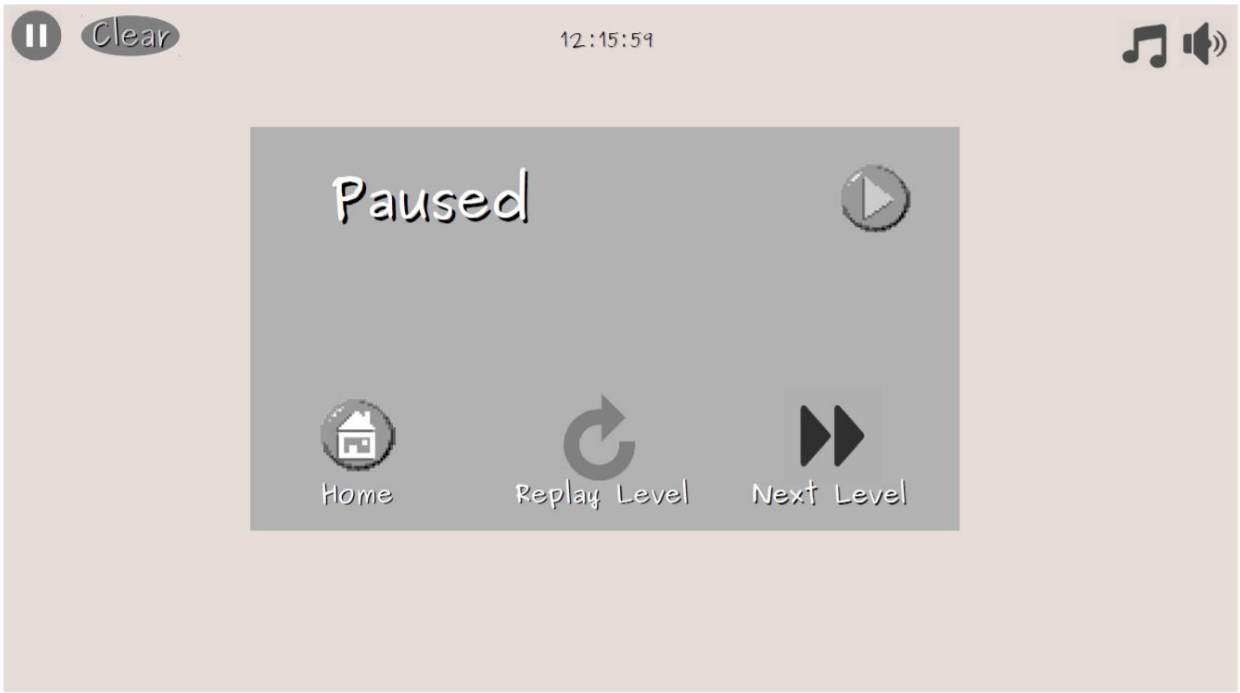

Figure 13 Pause Screen

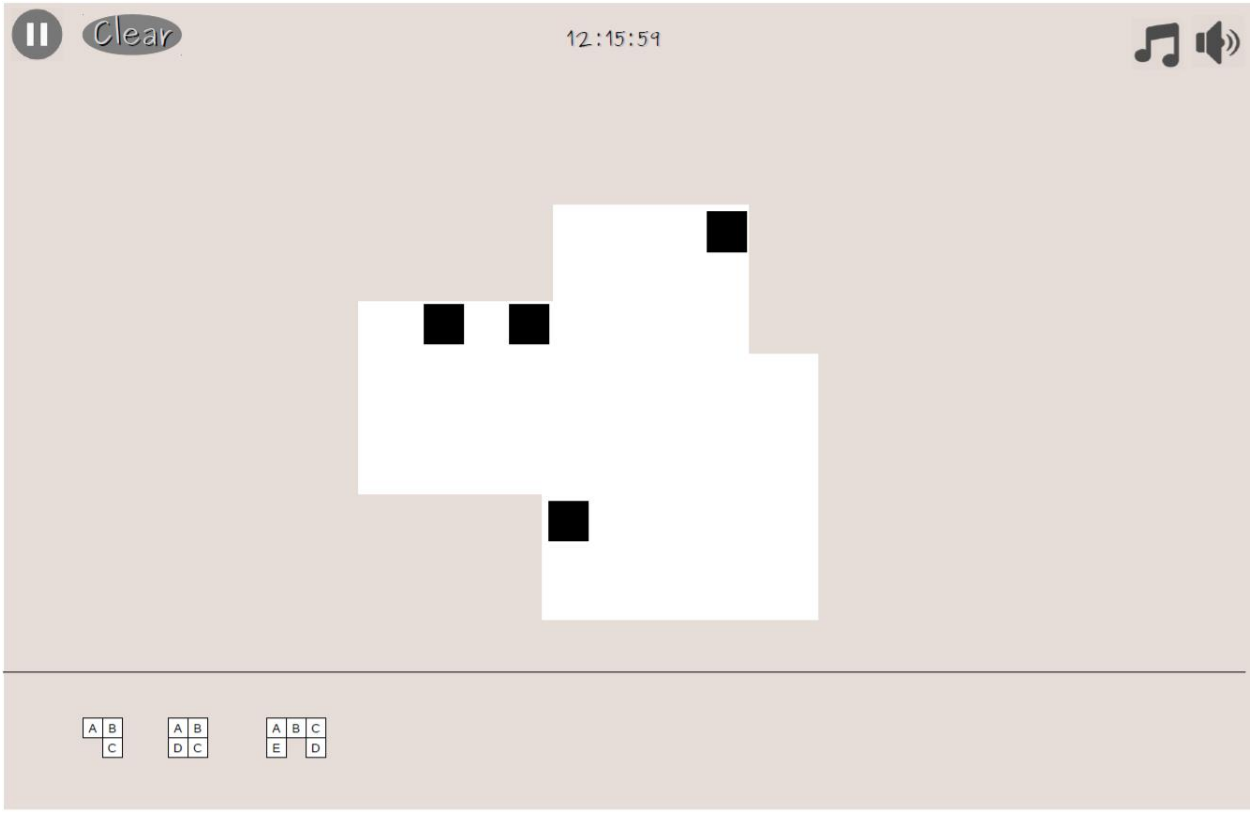

Figure 14 Puzzle Mode Screen

Figure 15 Level Menu for Puzzle Mode

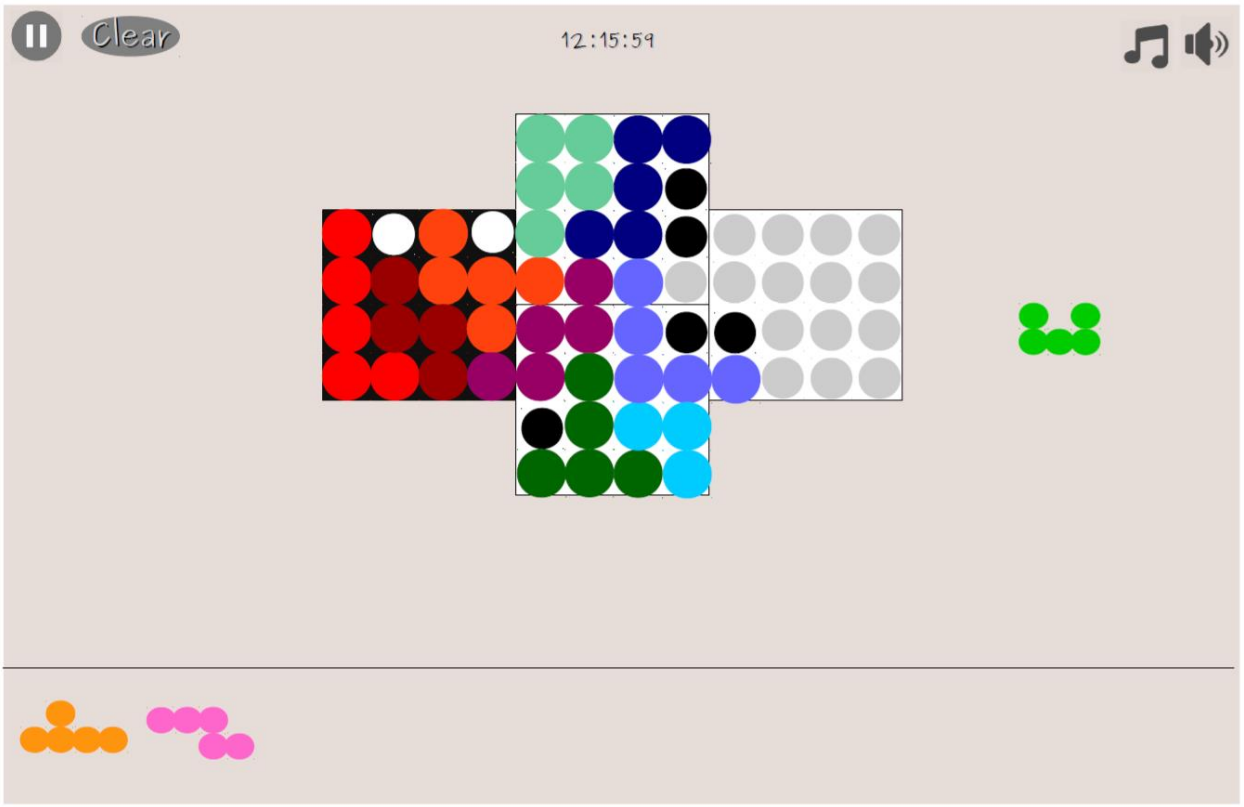

Figure 16 Vanilla Mode Screen

Figure 17 Credits

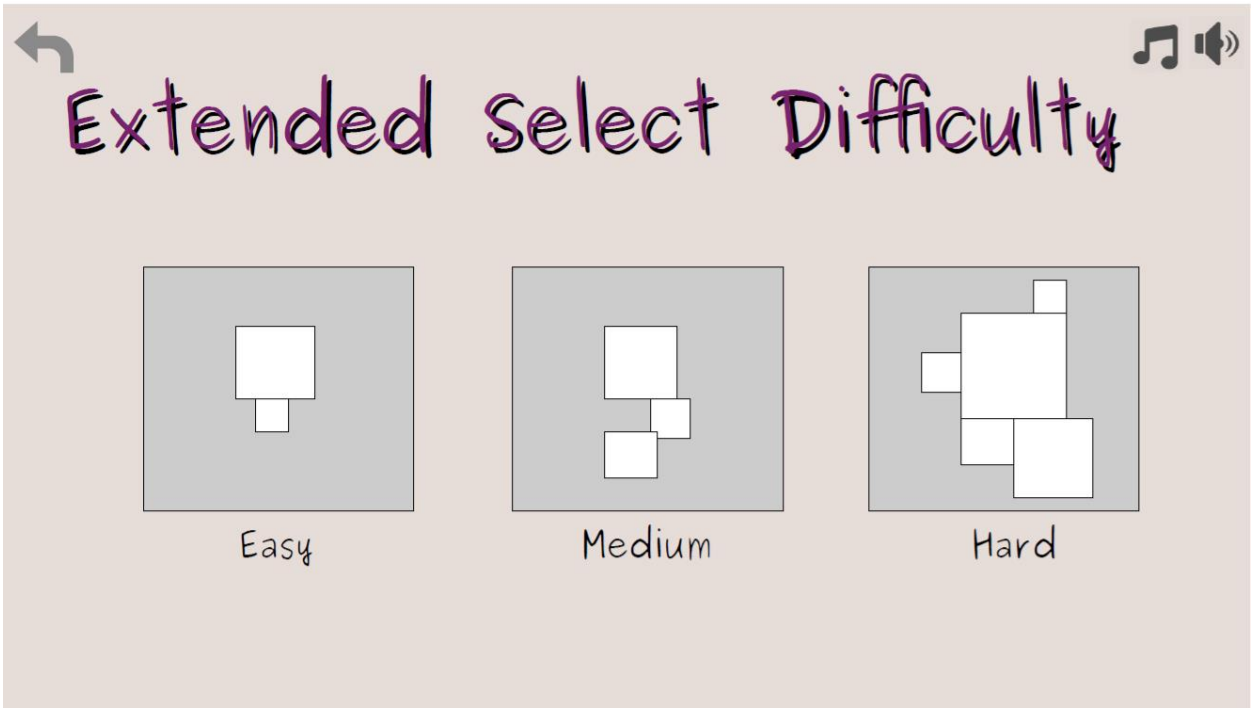

Figure 18 Select Difficulty Menu

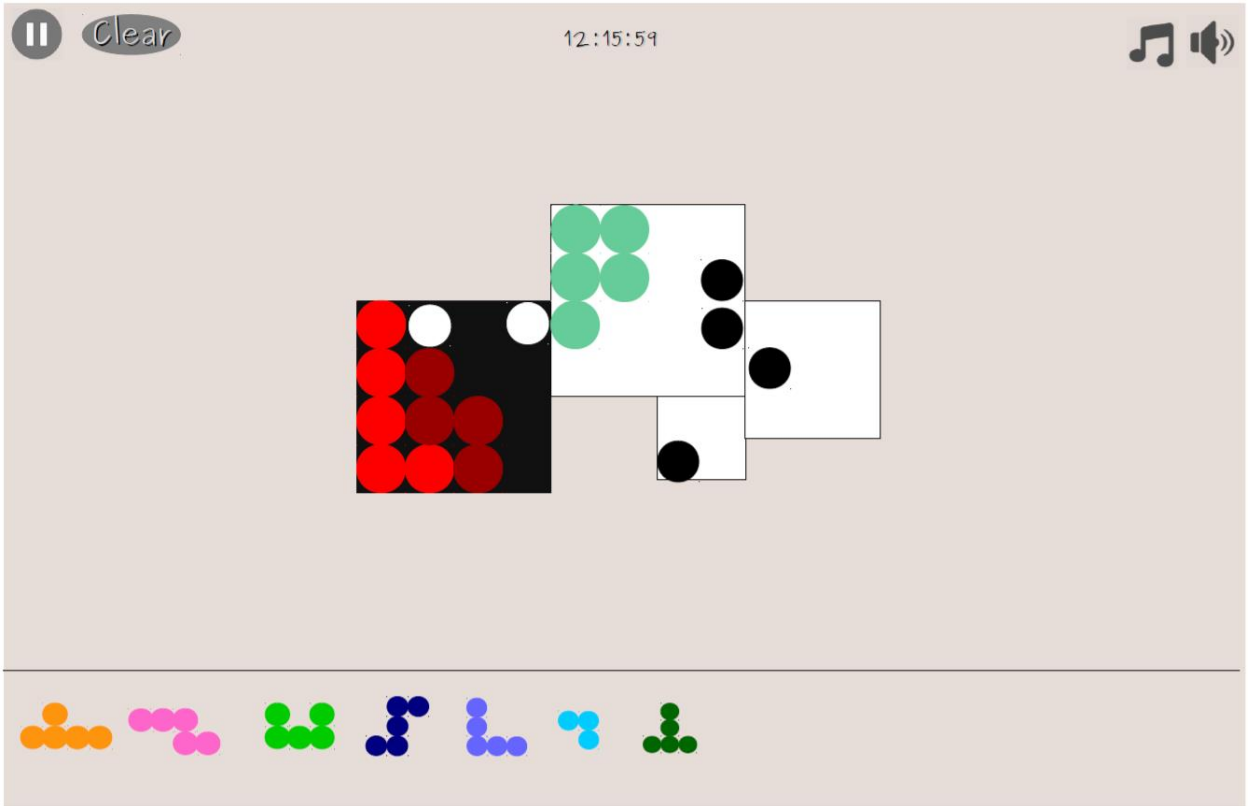

Figure 19 Extended Mode Screen

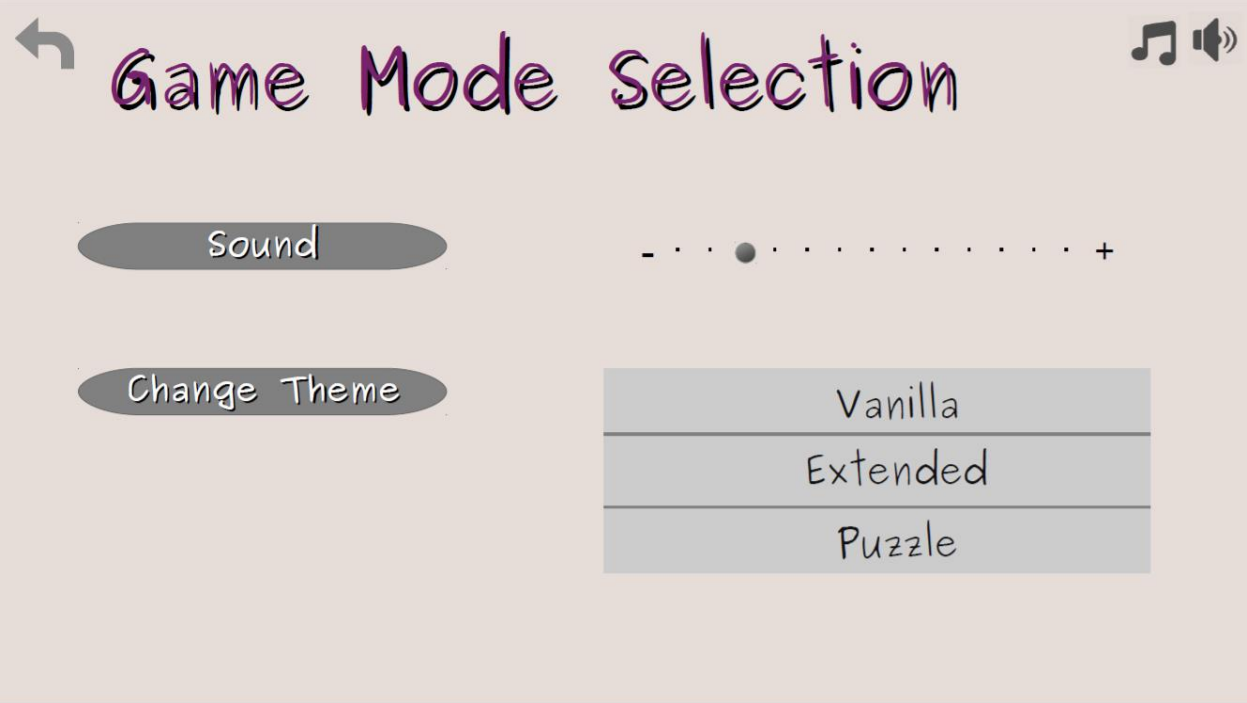

Figure 20 Settings

4 References

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

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

3) https://www.opensource.org/licenses/MIT