Bilkent University

Department of Computer Engineering

CS-319 Project

Quadrillion

Analysis Report

Group No: 1H
Group Name: COGENE

Osman Orhan Uysal

Samet Özcan

Mehmet Alper Karadağ

Ziya Erkoç

Talha Murathan Göktaş

Contents

Analysis Report

Quadrillion

1. Introduction

Quadrillion is a puzzle-board game in which the player needs to fill all the vacancies on the board using prefabricated pieces. Game consists of 4 sub-boards each of which can be combined in a lot of way to create the whole board. Each sub-board has two or three places which are blocked in a way that the marbles of the pieces cannot be placed there. Sub-boards are also double sided which indeed increases the combination possibilities.

In this project, we aimed to digitalize the Quadrillion game to make it more attractable and fun by adding features that it is not possible to add to the board game version.

2. Current System

Board version of the Quadrillion contains the pieces and 4 sub-boards. It allows users to attach sub-boards together to create different combinations. It contains a manual which includes around 60 challenges with solutions.

3. Overview

3.1. Piece

A piece is the main component of game which consists of marbles that are tied together and user can drag it around. User aims to find an available spot on the ground for each piece in order to complete the game. There are 12 different pieces in the game. A Piece can be rotated 90-degree clockwise, flipped along the y-axis and moved around.

3.2. Ground

A ground represents 4x4 sub-board where some locations are not available. Combination of these 4x4 sub-boards results in a level. Like Piece, Ground can be flipped around (front-side or back-side of the ground can be used), moved around and rotate 90-degree clockwise.

3.3. Level

Level is a combination of 4 grounds. In our game, there are predefined levels which already exist in the manual of the board version as well as user created levels.

3.4. Hint

If user stucks at some point there will be hint provided for him/her. Hint basically reveal the location of one of the pieces. User will be given some amount of hint tokens which will decrease as user requests hint. By paying small amount of fee user will be able to get extra hints. User should use their hints wisely since at some point they may be scarce.

4. Functional Requirements

4.1. Play Game

User can play either ranked game or casual game.

4.1.1. Casual Game

In this mode, user selects either one of the predefined levels or user created levels. User drags pieces and drops them on the grounds in order to fill all the vacancies on the grounds. In the meantime, number of moves he/she made and time elapsed for the completion of the level is tracked so that user can compete with himself/herself. In case user gets stuck, hint will be provided to him/her if he/she has enough hints available. Otherwise, he/she can buy hints. User's performance in casual game is not added to the leaderboard of the level.

4.1.2. Ranked Game

Unlike casual mode, in ranked mode, the user will be randomly assigned to a level. Again, user can drag & drop the pieces to complete the level. However, he\she will be able to neither request a hint. If user successfully completes the level, based on the time it took to solve the level and the number of moves he made, he will be ranked on the leaderboard of the level.

4.2. Compose Level

User can create levels by moving, flipping and rotating the grounds. Then he/she can hit the submit button so that the level can be checked for validity. The levels are accepted after checking if the level is a valid combination of grounds, has a valid solution and this combination has not been created before.

4.3. Select Level

Both predefined levels and user defined levels are displayed on level selection screen. User can see the leaderboard of the level before playing it.

4.4. Buy/Use Hint

Users are given the ability to use and buy hints to proceed through the game. When user is stuck at a level in casual mode, he/she can use one of his/her hints to reveal the position of one of the pieces. Users are given some amount of hints initially and they are expected to use them wisely; that is, they should not waste them. If they run out of hints, they can buy some by entering their credit card information.

5. Non-functional Requirements

5.1. Usability

User can easily navigate through the user interface of Quadrillion since it is very simple and self explanatory. If user does not know how to play the game s/he can take a look at instructions window in game. Moreover, instructions for composing a new level are present in the compose level screen.

5.2. Reliability

Quadrillion, as its name suggests, has huge amount of possibilities in terms of ground and piece combinations. Unfortunately, not every ground combination has a solution and it is not easy to realize it on board version. However, digital version of the quadrillion ensures that every level user sees on level selection screen has at least one solution even the user created ones. Furthermore, users can safely buy hints to get a helping hand when they are stuck in a level. These hints are given in the most beneficial way for the user.

5.3. Performance

The only performance issue of the game is related with network speed of the user. That is, uploading a new level, authorization, buying hints or uploading scores to the leaderboard is dependent on users network connection quality. Other than that our game is lightweight and has no system requirements.

5.4. Supportability

The digital version of the Quadrillion can run on any machine which has Java installed regardless of the operating system (Windows, Mac OS, Linux) thanks to the power of JVM (Java Virtual Machine).

6. System Models

6.1. Use-Case Model

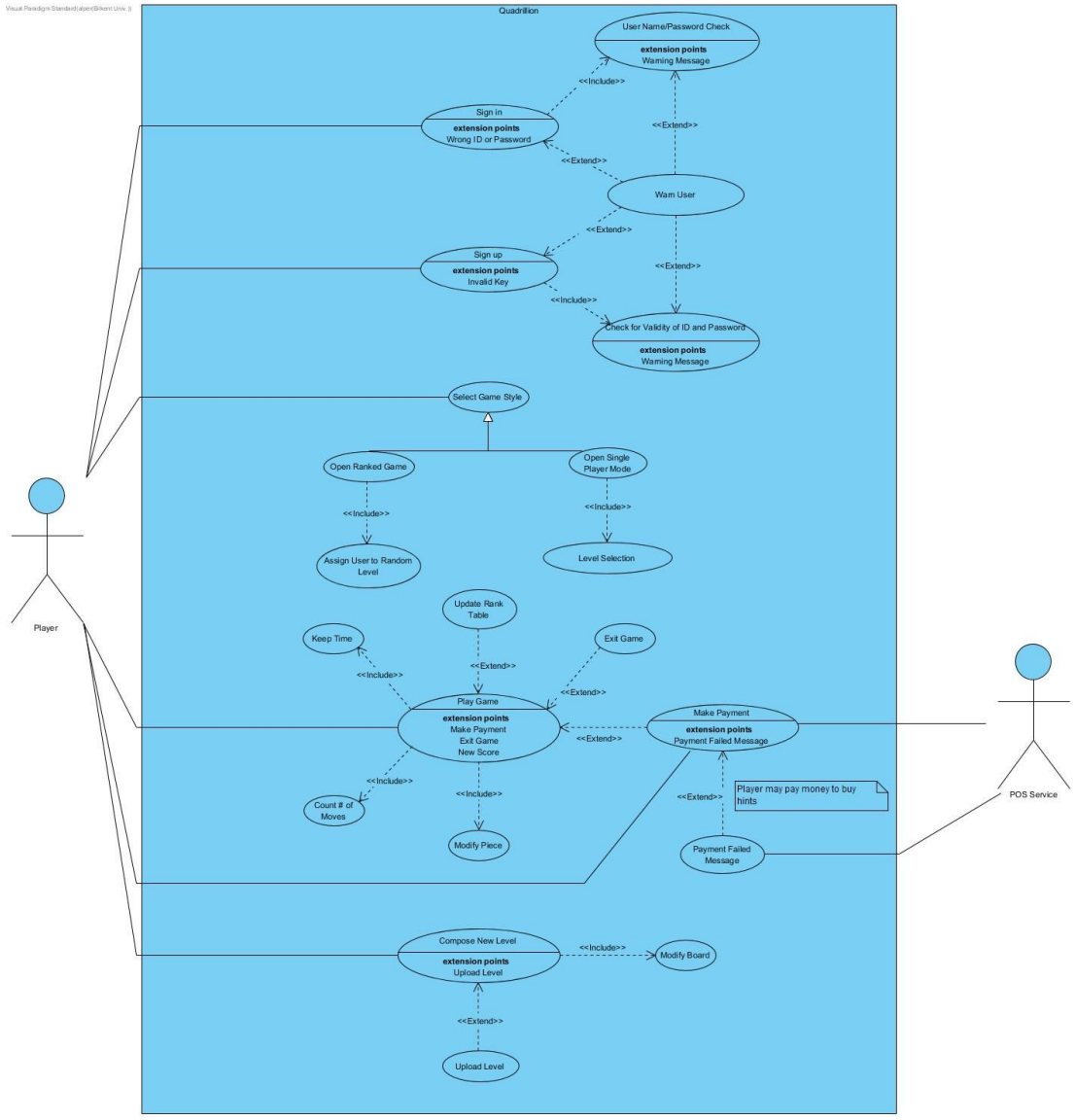

Figure 1: Use Case Diagram

Use case name **Sign in**

Participating actor

●Player

Entry condition

●Player pushes the sign in button

Flow of Events

●Player enters his/her id

●Player enters his/her password

Exit condition

●Player sign in to the system successfully or enters wrong id or password

Special requirement

●Only 3 attempts are allowed in a minute

Use case name **Sign up**

Participating actor

●Player

Entry condition

●Player pushes the sign up button

Flow of Events

●Player enters id

●Player enters password

Exit condition

●Player sign up to the system successfully or enters invalid id or password

---

Use case name **Warn User**

Entry condition

●Player enters invalid id or password

Flow of Events

●Warning message is displayed

Exit condition

●Player notices the notification and another try might be provided

---

Use case name **User Name/Password Check**

Entry condition

●Player tries to sign ip

Flow of Events

●Whether the entered id and password are matched with one of the ones in the memory is checked

Exit condition

●Check process is completed

Use case name **Check for the Validity of ID and Password**

Entry condition

●Player tries to sign up to the system by entering an id and a password

Flow of Events

●Whether the entered id and password are new or not is checked

Exit condition

●Check process is completed

Use case name **Select Game Style**

Participating actor

●Player

Entry condition

●Player pushes to the ‘Select Game Style’ button on the main menu

Flow of events

●Player selects either ranked game or single player mode

Exit condition

●The selection is done

Special requirement

●POS service is included in the use case only if player tries to lock one of the unlocked levels

---

Use case name **Open Ranked Game**

Entry condition

●Player pushes to the ‘Open Ranked Game’ button during the selection of game style

---

Use case name **Assign User to Random Level Report**

Entry condition

●Ranked Game Style is selected

Exit condition

●Player is assigned to a random level

Use case name **Open Single Player Mode**

Entry condition

●Player pushes to the Open Single Player Mode button during the selection of game style

---

Use case name **Level Selection**

Participating actors

●Player

Entry condition

●Single Player Mode is selected

Flow of Events

●Levels are displayed

Exit condition

●Either locked or unlocked level is selected

Use case name **Make Payment**

Participating actors

●Player, POS Service

Entry condition

●Player wants to get hint(s) during the game or on the main menu

Flow of Events

●Player selects how many hints is s/he going to purchase

●Player enters the information required for the purchase process

●Player approves the purchase process

Exit condition

●Hins are purchased successfully

Use case name **Payment Failed Message**

Participating actors

●POS Service

Entry condition

●Player attempts to buy hints but enters invalid information for payment or some interruption occurs

Flow of Events

●The message to inform the player about failure of payment is displayed

Exit condition

●Player either makes another try or gives up

Use case name **Play Game**

Participating actor

●Player

Entry condition

●Player pushes to the 'Play a Game' button on the main menu

Flow of events

●Player plays the game

Exit condition

●Either player places all pieces successfully or he/she gives up

---

Use case name **Modify Piece**

Participating actor

●Player

Entry condition

●Player starts to play a game

Flow of events

●Player tries to place all pieces on the board to fill the board completely by selecting, dragging, flipping or rotating the pieces

Exit condition

●Player makes his/her try

Use case name **Exit Game**

Participating actor

●Player

Entry condition

●Player gives up the game

Flow of events

●Player exits the game

Exit condition

●The game is ended without success.

●User is redirected to main menu

Use case name **Keep Time**

Entry condition

●The game starts

Flow of events

●The time passed since the start of the game is counted and displayed on the game screen

Exit condition

●The game ends anyway

Special requirements

●The time is paused during the payment process

Use case name **Count Number of Moves**

Entry condition

●Player makes a try

Flow of events

●Number of moves made is increased by one and displayed on the game screen

Exit condition

●The game ends anyway

Use case name **Update Rank Table**

Entry condition

●Player finished the level successfully in a ranked game mode and achieves to enter to the Rank Table

Flow of events

●Player's name and his/her score is placed on the Rank Table and the last score and its owner is dropped from the table

Exit condition

●The Rank Table is updated

Use case name **Compose New Level**

Participating actor

●Player

Entry condition

●Player pushes to the ‘Compose New Level’ button on the main menu

Flow of Events

●Player composes a new level

Exit condition

●Player attempts to upload the level or gives up

Use case name **Modify Board**

Participating actor

●Player

Entry condition

●Player starts to compose a new level

Flow of Events

●Player tries to create a complete new level by selecting, dragging, flipping, rotating and placing all boards

Exit condition

●Player attempts to upload the level or gives up

Use case name **Upload Level**

Participating actor

●Player

Entry condition

●Player tries to upload the level that s/he created

Flow of Events

●Whether the created level is completed or not is checked and added to the ranked game mode if it is approved

Exit condition

●The created level is either added to the ranked game mode or rejected

6.2. Object and Class Model

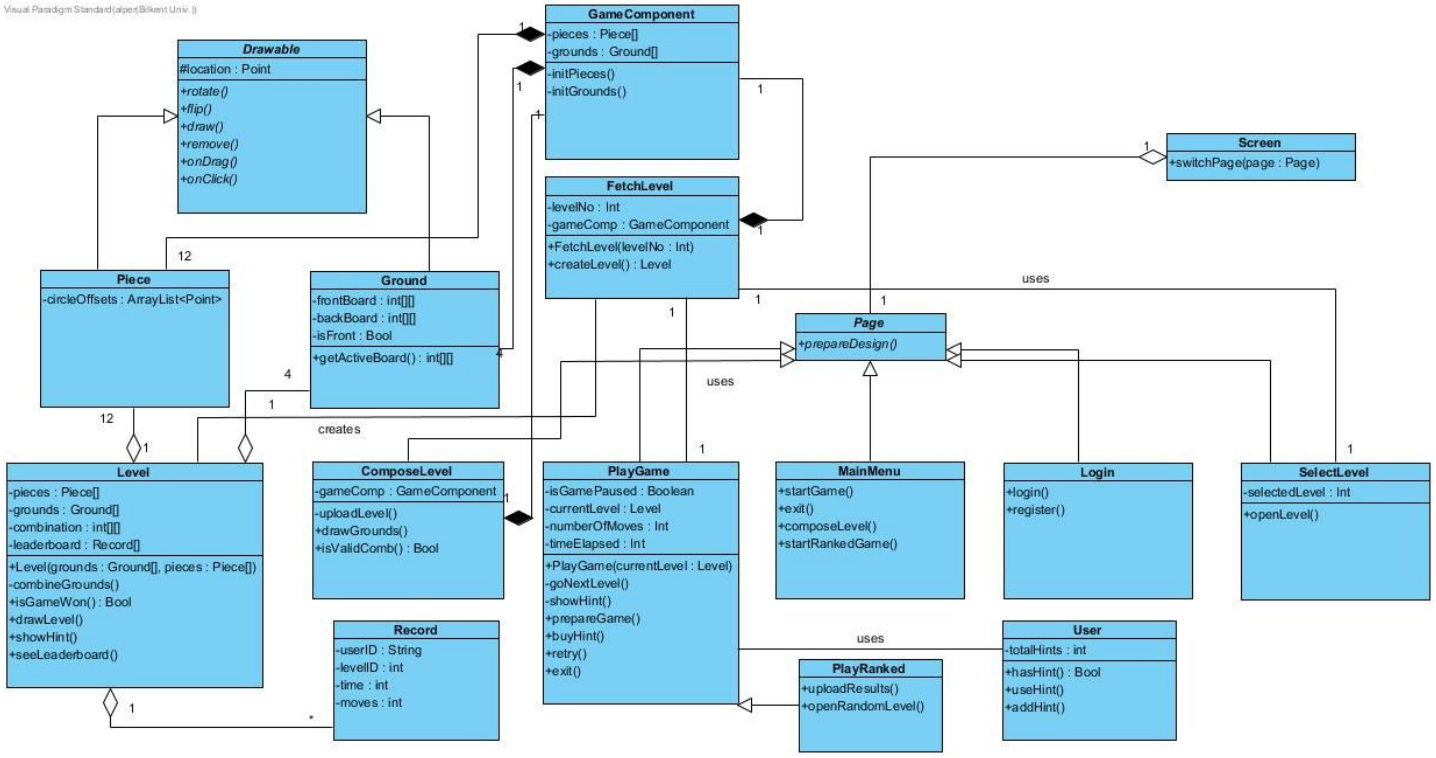

Figure 2: Class Diagram

6.2.1. Drawable Class

This class is defined to represent the main components of the game; namely, Piece and Ground which are subclasses. It has only one attribute.

1. location : Location of the component on the screen.

6.2.1.1. Piece Class

This represents pieces of the Quadrillion game which consist of marbles that are tied to each other in various ways. This class has one attribute:

1. circleOffsets - It holds the information of how marbles are positioned thereby defining the structure of the piece.

6.2.1.2. Ground Class

This class represents sub-board (there are 4 of them in Quadrillion). Each sub-board has two faces which have different occupied* places. There are three attributes:

1. frontBoard - It holds the information of which cells of the front side of sub-board is occupied.
2. backBoard - It holds the information of which cells of the back side of sub-board is occupied.
3. isFront - It checks whether front face is used or not.

* “occupied place” refers to a cell on the sub-board being closed so that no marble can be put there.

6.2.2. Level Class

This class is responsible for holding information about the level such as how sub-boards are oriented and how they are connected to each other. It has 4 attributes:

1. pieces - The pieces that will be used in the level.
2. grounds - The grounds that will be used in the level.
3. combination - The matrix that will be built by using grounds (sub-boards). It brings all sub-boards together an acts as a single board.
4. leaderboard - It is an array of records and it holds information about the best performers of the level.

6.2.3. GameComponent Class

Since pieces and grounds are all prefabricated, this class will be responsible for fabricating them. It holds *pieces* and *grounds* it fabricated as attributes.

6.2.4. Page Class

This class holds the layout of the user interfaces. In the project, there are 6 scenes that are subclasses of this class and that user can interact with.

6.2.4.1. PlayGame Class

User actually plays the game here. This class has four attributes:

1. isGamePaused - It checks whether game is paused.
2. currentLevel - It holds the level currently being played.
3. timeElapsed - It counts the time elapsed until sloving the puzzle.
4. numberOfMoves - It counts the number of times player makes a move.

6.2.4.1.1. PlayRanked Class

This class extends PlayGamce class. User can also play the game competitively. numberOfMoves and timeElapsed are uploaded in order to rank the user.

6.2.4.2. MainMenu Class

This class represents the Main Menu where user can start or exit the game.

6.2.4.3. ComposeLevel Class

This class represents the level composition page. By merging predefined sub-boards in various ways, users can create new levels and upload it. It has single attribute:

1. gameComp: It will be used to draw Grounds (sub-boards).

6.2.4.4. Login Class

Users can register and login to the system through this page.

6.2.4.5. SelectLevel Class

This class is responsible for selection of level to be played. It holds the level selected by the user as an attribute; namely, *selectedLevel*.

6.2.5. FetchLevel Class

This class fetches the information of a level from database or a text file and creates a new level. It has two attributes:

1. levelNo - Id of the level that will be fetched

2. gameComp - It holds the copies of game componentes that are Pieces and Grounds and will be used to create level.

6.2.6. Screen Class

This class is responsible for creating main window of the game and switching between pages.

6.2.7. User Class

This class holds number of hints available for the user and regulates the hint addition and subtraction.

6.2.8. Record Class

This class represents records in the leaderboard. It holds how many moves user made and how much time it took for him/her to solve the level as well as unique user number and level number.

6.3. Dynamic Models

6.3.1. Sequence Diagram

6.3.1.1. Login

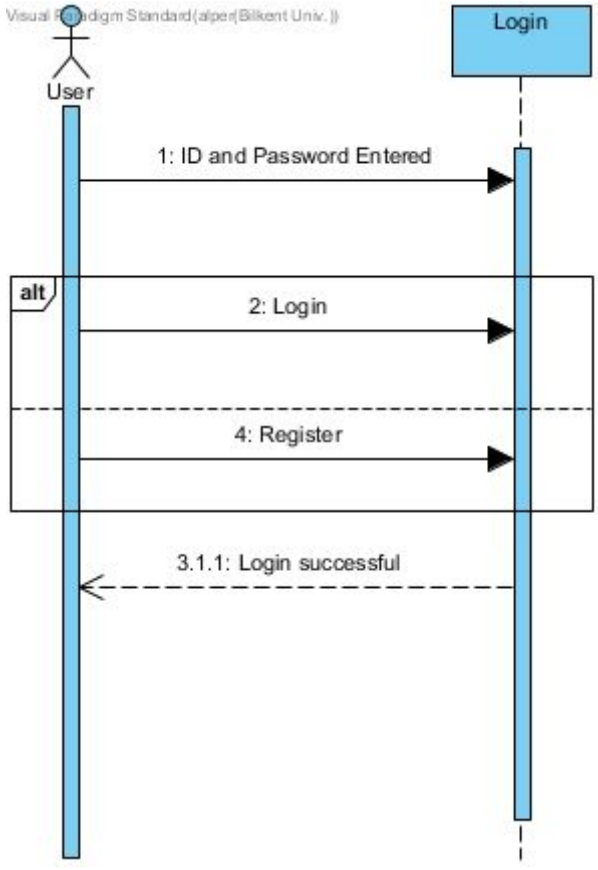

Figure 3: Sequence diagram for login

Login screen is the first screen that user sees when he/she opens the game. In this screen user should choose a user id and password to register to the system. If user already has an account he\she can directly log in to the system.

6.3.1.2. Play Casual

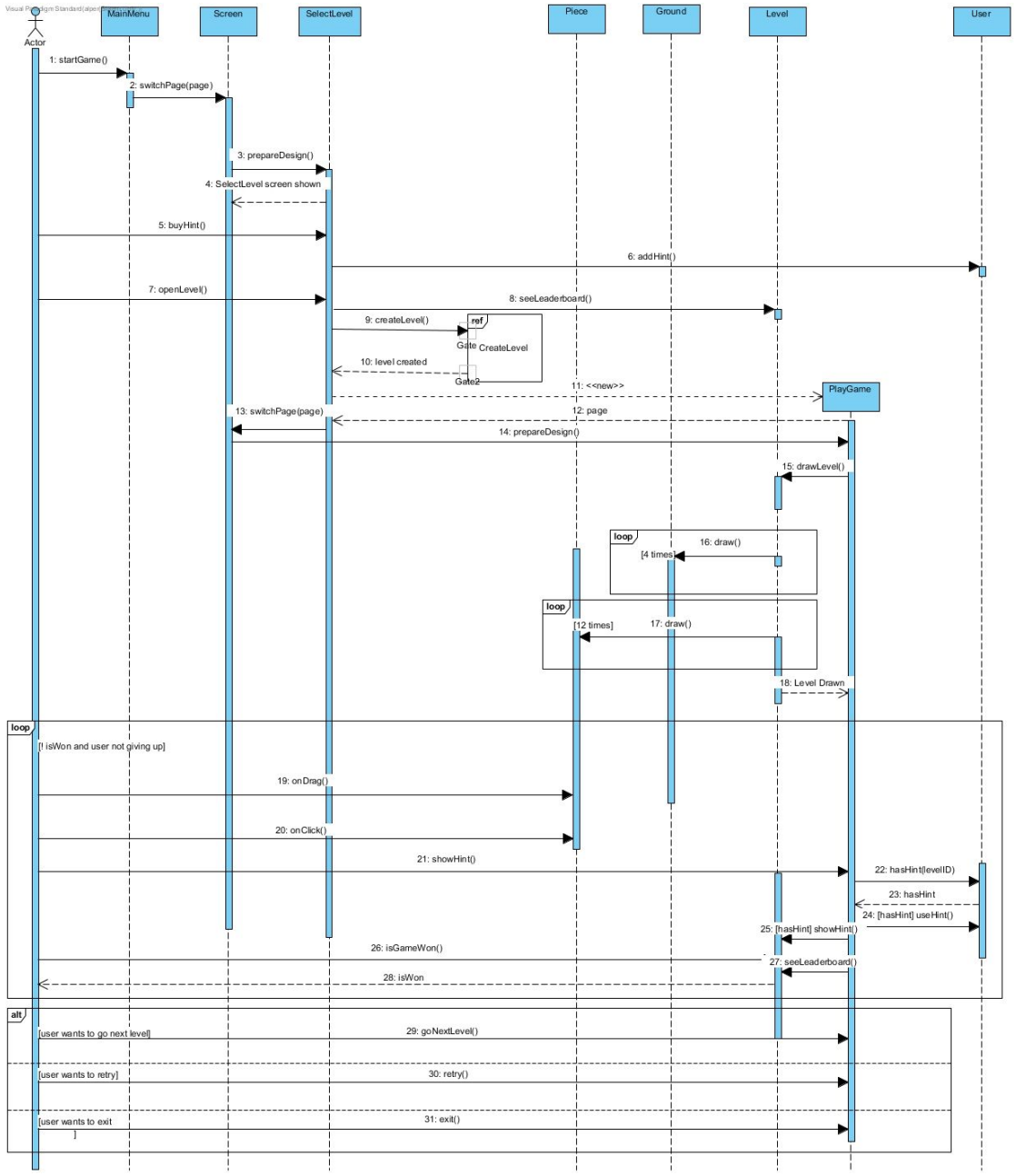

Figure 4: Sequence diagram for Casual game

User starts playing single player game and level selection screen shows up in front of him/her. If user needs some hints, he/she buys it through PlayGame class. Then, PlayGame class communicates with User class to increase the user's number of hints. Having bought hint, he/she opens a level using Select Level class. Select Level class then creates level by communication with CreateLevel sequence diagram. Created level is returned back to the Select Level class. Select Level class creates new PlayGame class that is the environment to play the game and sends it to Screen class to show it on the screen. Screen class says PlayGame class to prepare itself for the gameplay. PlayGame class commands all Pieces and Grounds to be drawn on the

screen and ready to be played. Then, user constantly drags and clicks pieces to complete the level. He/she can use hints. Whether or not he completed the level is controlled repeatedly. If he completes or gives up, leaderboard is shown to him. Finally, he can either retry or proceed to the next level.

6.3.1.3. Play Ranked

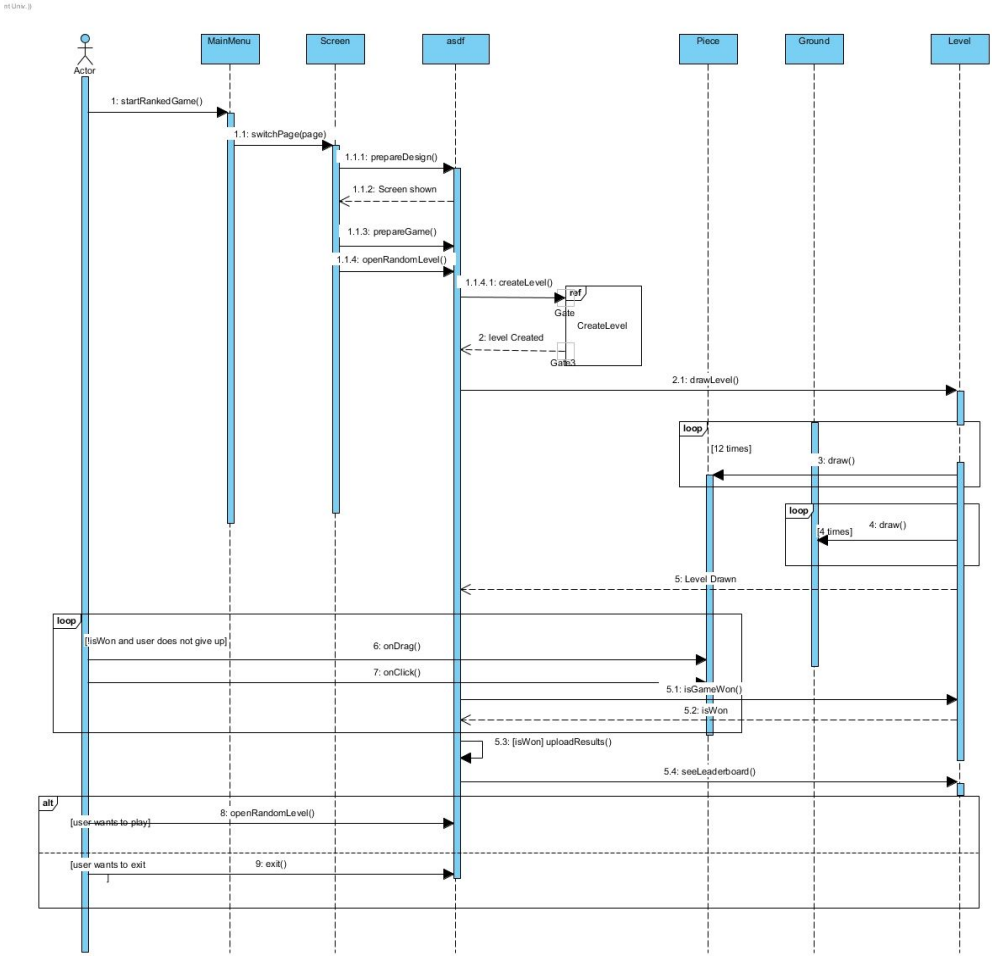

Figure 5: Sequence diagram for Ranked Game

User starts ranked game through MainMenu class. MainMenu tells Screen class to show Ranked Game screen. Screen induces PlayRanked class to prepare itself and open a random level since user will be ranked based on diperformance. Then, PlayRanked class creates level through Create Level sequence diagram. After that, PlayRanked class tells all grounds and draws to be drawn on the screen. Unless user gives up or completes the level, he/she can move pieces to complete the level. Constantly game board is checked to control whether level is completed. If level is completed, then user results are uploaded for ranking purposes. Finally, user sees the leaderboard and then he/she either plays new level or exits.

6.3.1.4. Create Level

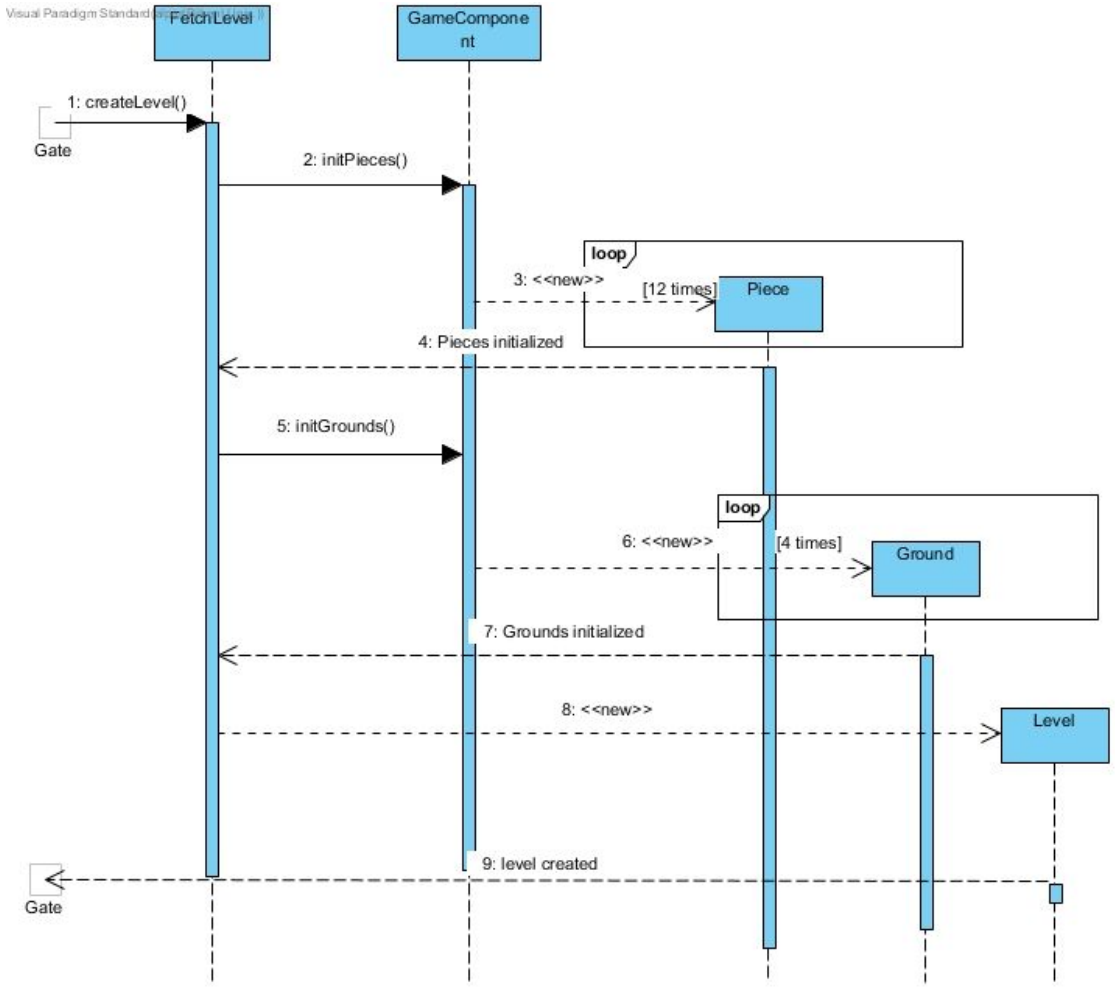

Figure 6: Sequence diagram for Create Level

When the level to be played is determined createLevel method of the FetchLevel class is called. Then initPieces and initGrounds methods of the GameComponent class is called so that 12 pieces and 4 grounds of the corresponding level can be created. After all the pieces and grounds are created an instance of Level class is created with given pieces and grounds. Finally created level is returned to the caller of createLevel.

6.3.1.5. Compose Level

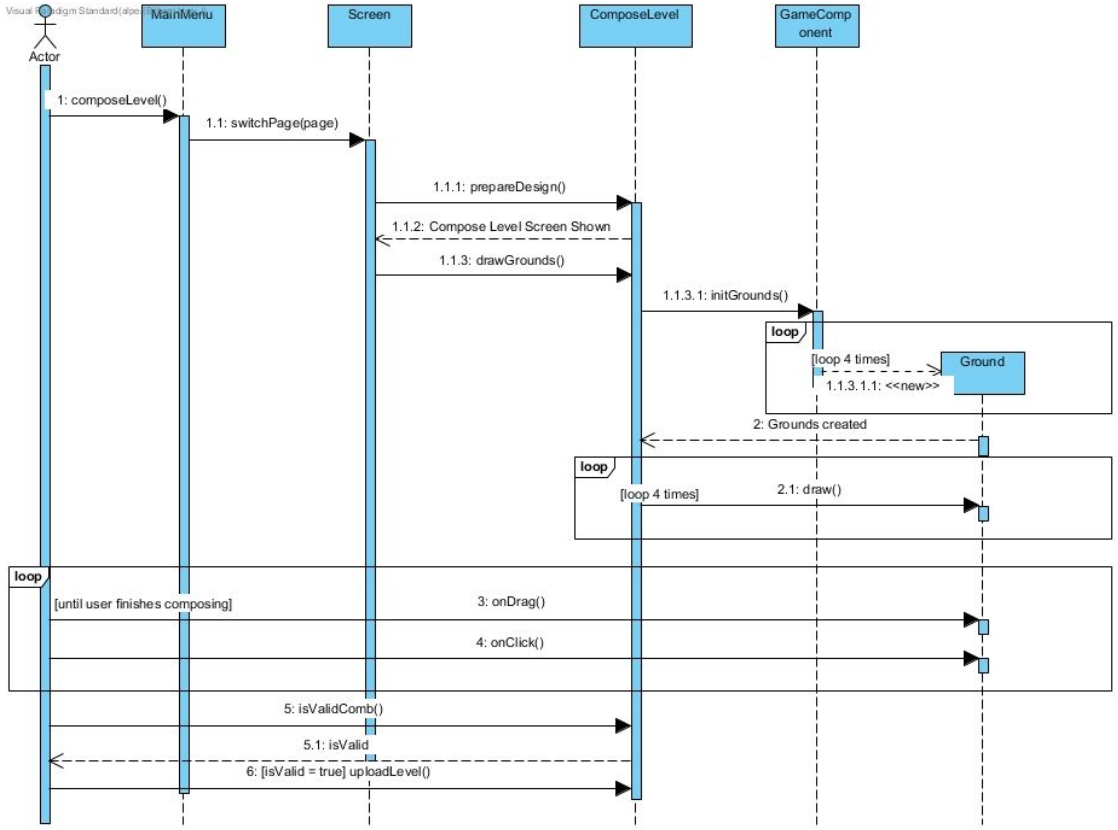

Figure 7: Sequence diagram for Compose Level

User can access compose level screen through main menu by pressing Compose Level button. Pressing the button triggers the switchPage method of Screen object. Then Screen object requests the specific design of the compose level screen by prepareDesign method. After changing the screen grounds that will be shown in the screen are requested from ComposeLevel class. With the help of GameComponent's initGrounds method 4 grounds that every level must have are created. Then these grounds are drawn to the screen with a loop. When the compose level screen is set, user can try new combinations with grounds by dragging them on screen and clicking to flip it. Finally, user can click on submit button which will first check the validity of the composed level then check if the level already exists. If everything is alright the level will be added to the game.

6.3.2. State Diagram

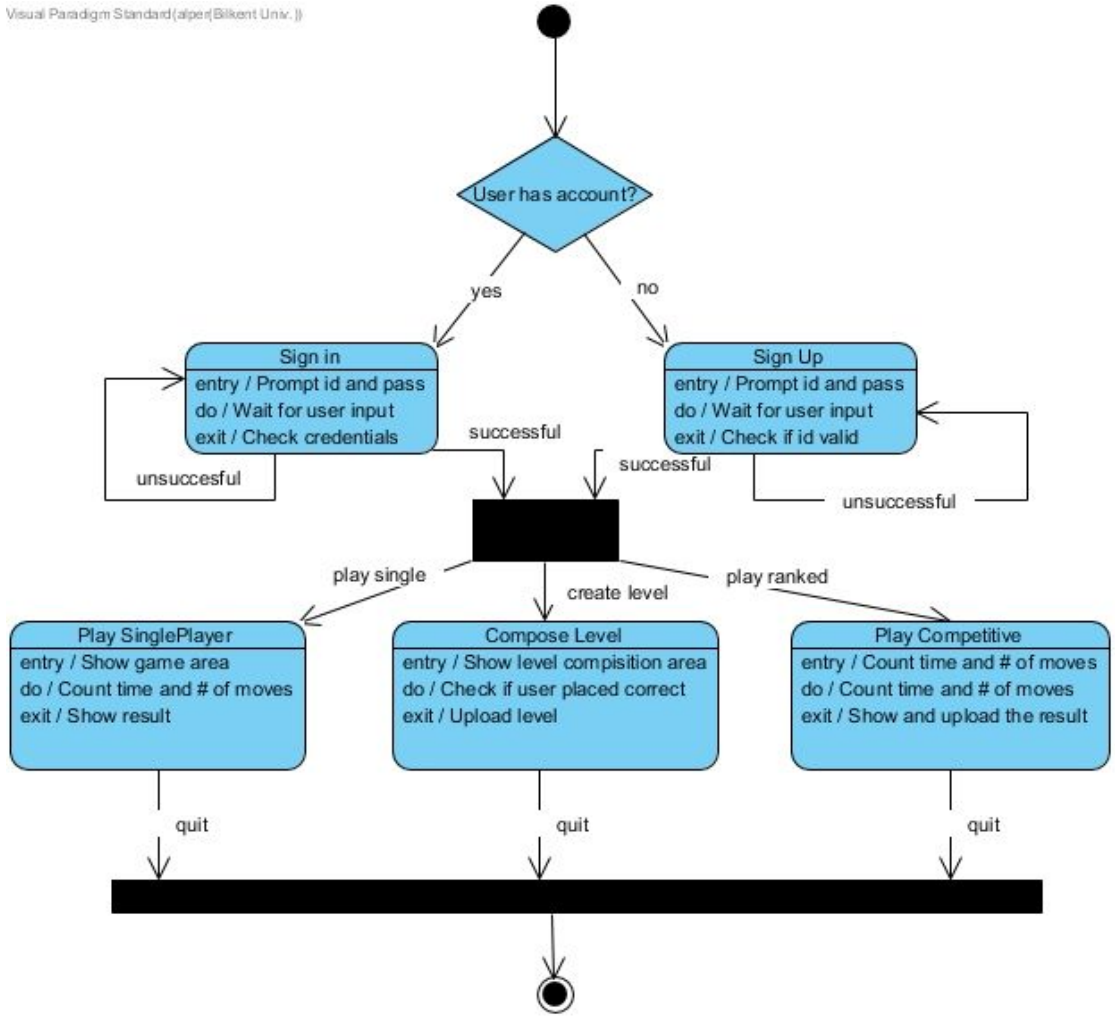

Figure 8: State diagram for Login Screen and Main Menu

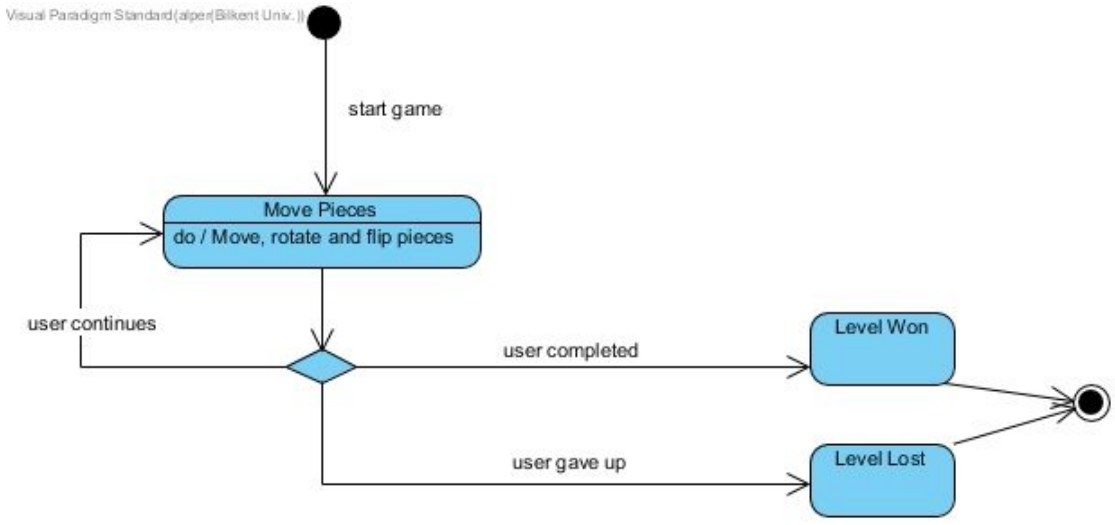

Figure 9: State diagram for Winning and Losing condition

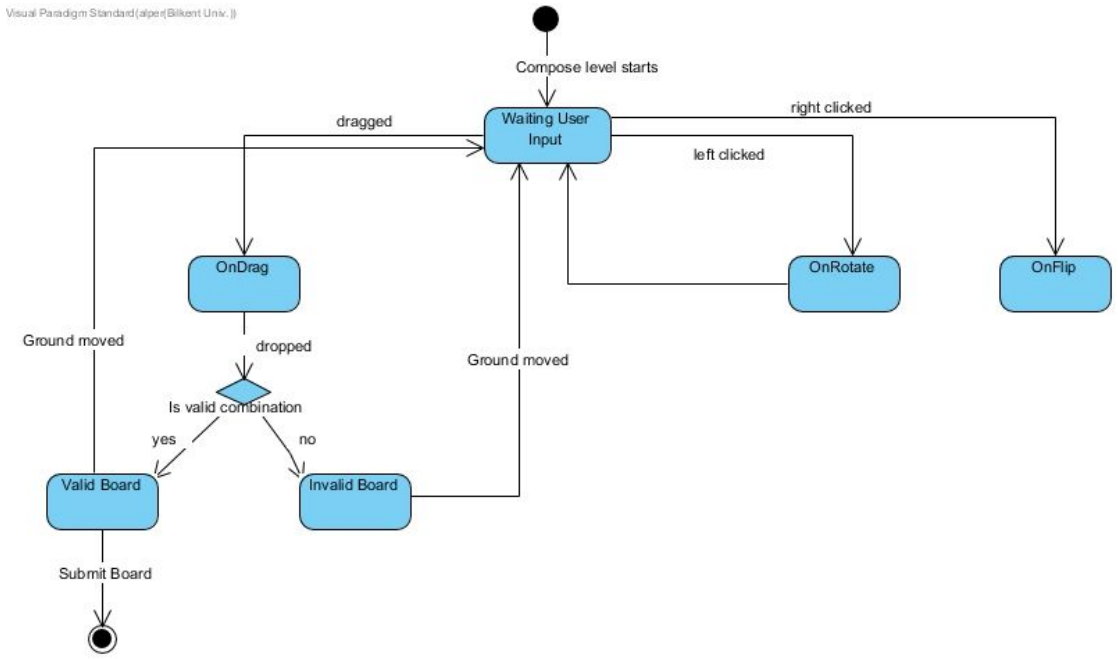

Figure 10: State diagram for Ground objects in Compose Level

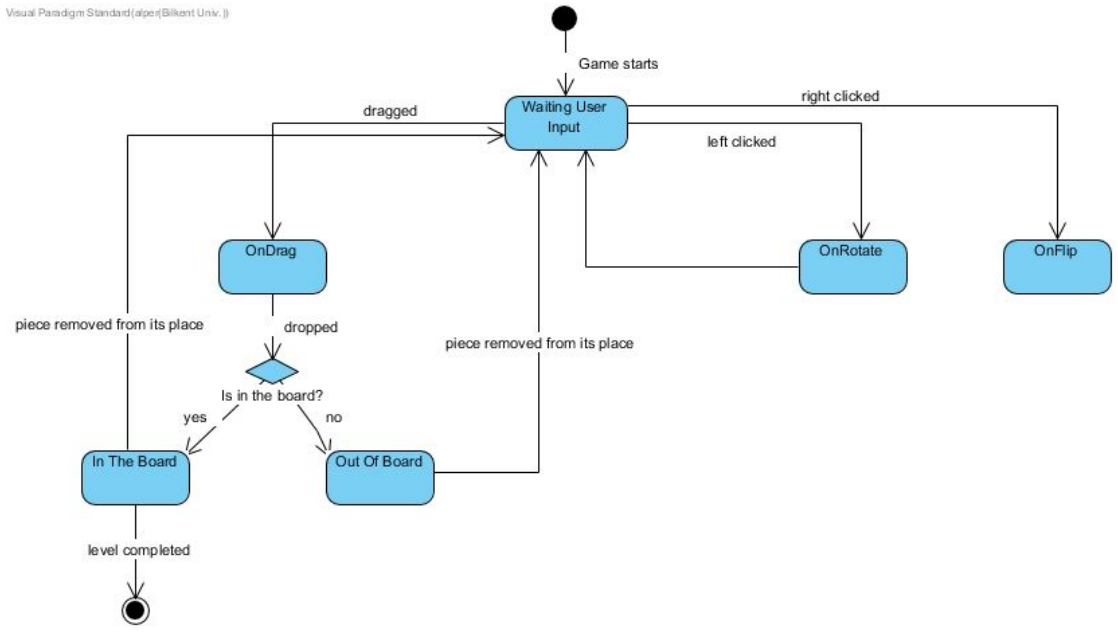

Figure 11: State diagram for Piece objects in game

6.3.3. Activity Diagram

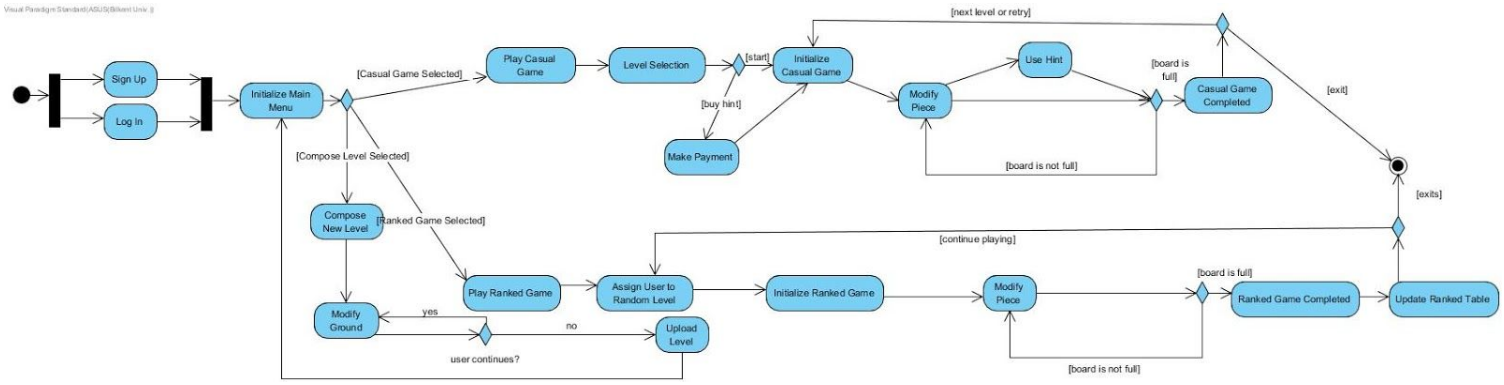

Figure 12: Activity diagram

6.3.4. User Interface

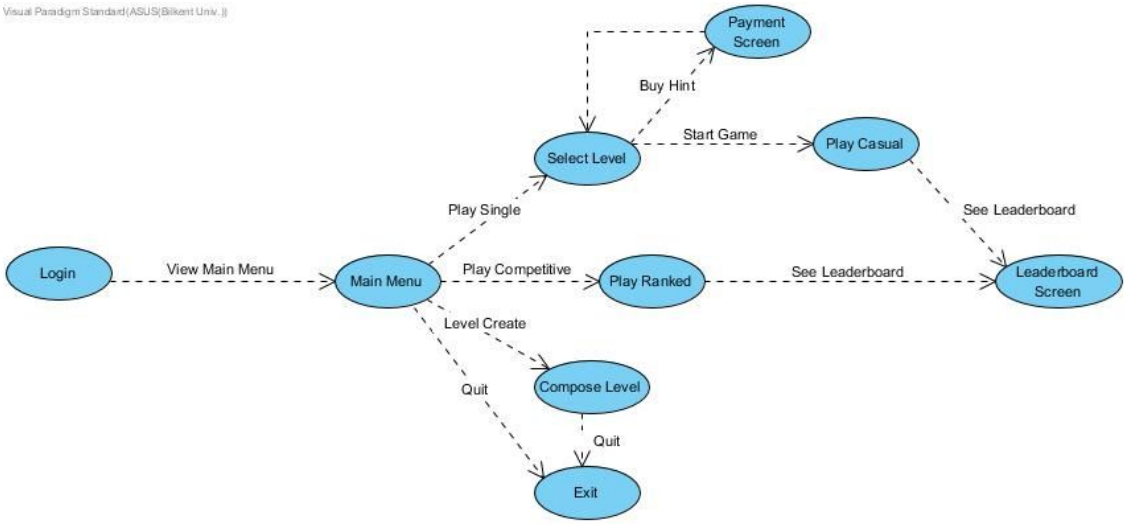

Figure 13: Navigational Path

Figure 14: Login Screen

Figure 15: Main Menu Screen

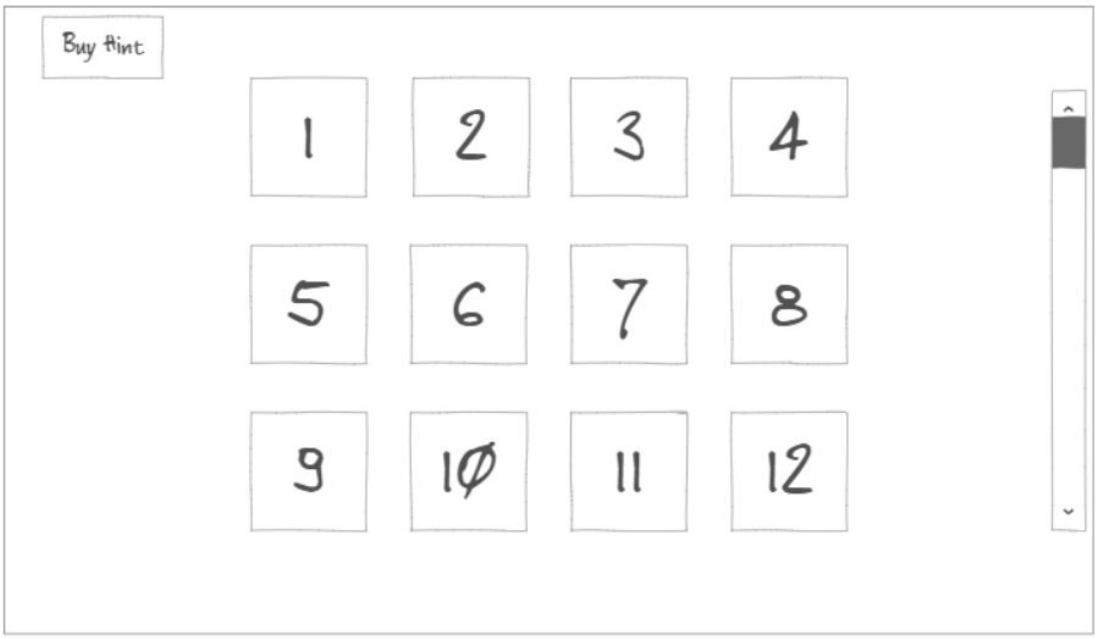

Figure 16: Select Level Screen

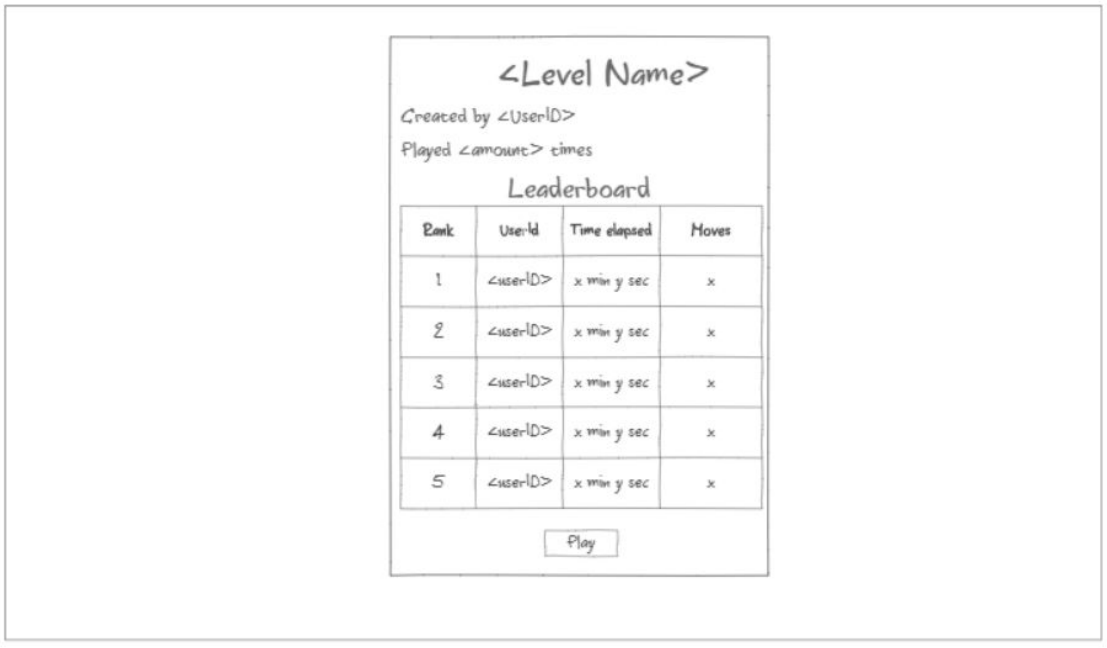

Figure 17: Leaderboard Screen

Figure 18: Hint Purchase Screen

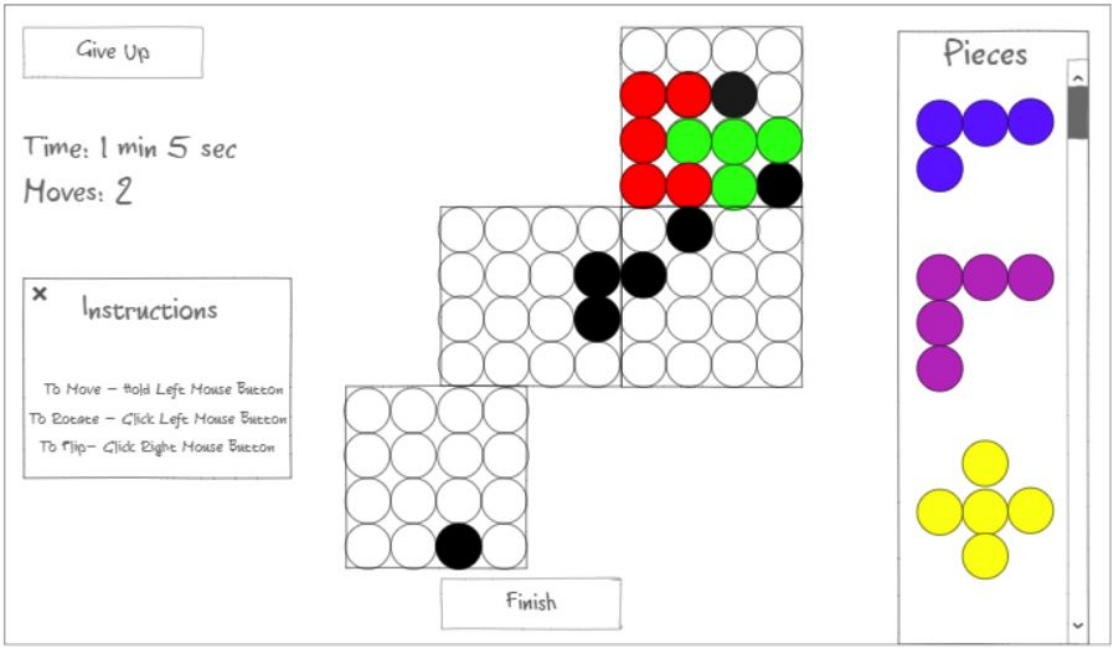

Figure 19: Play Casual Screen

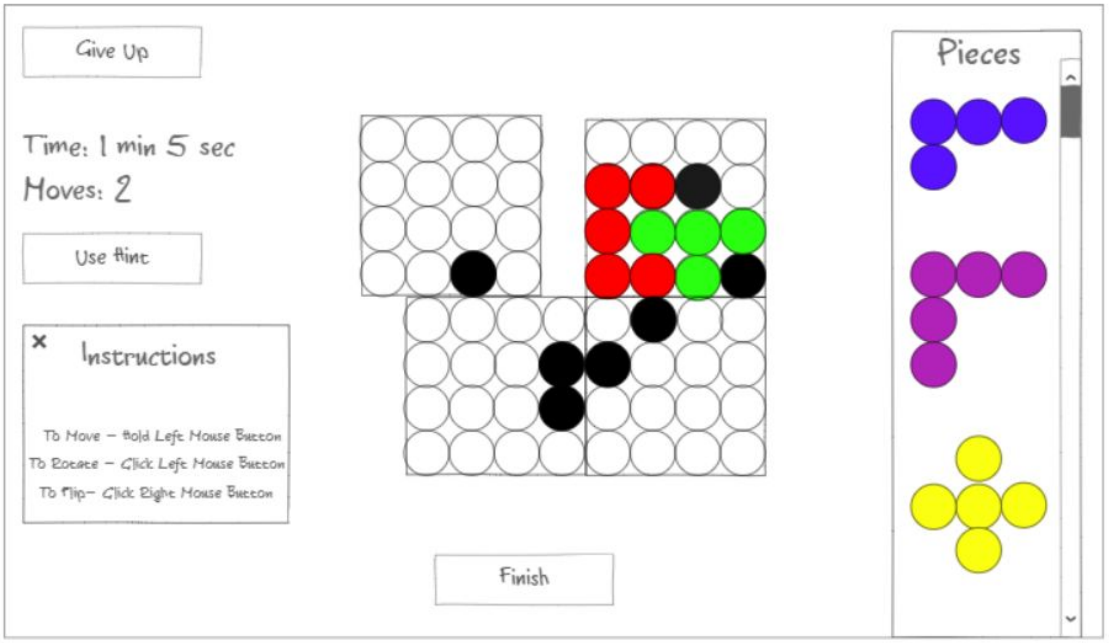

Figure 20: Play Ranked Screen

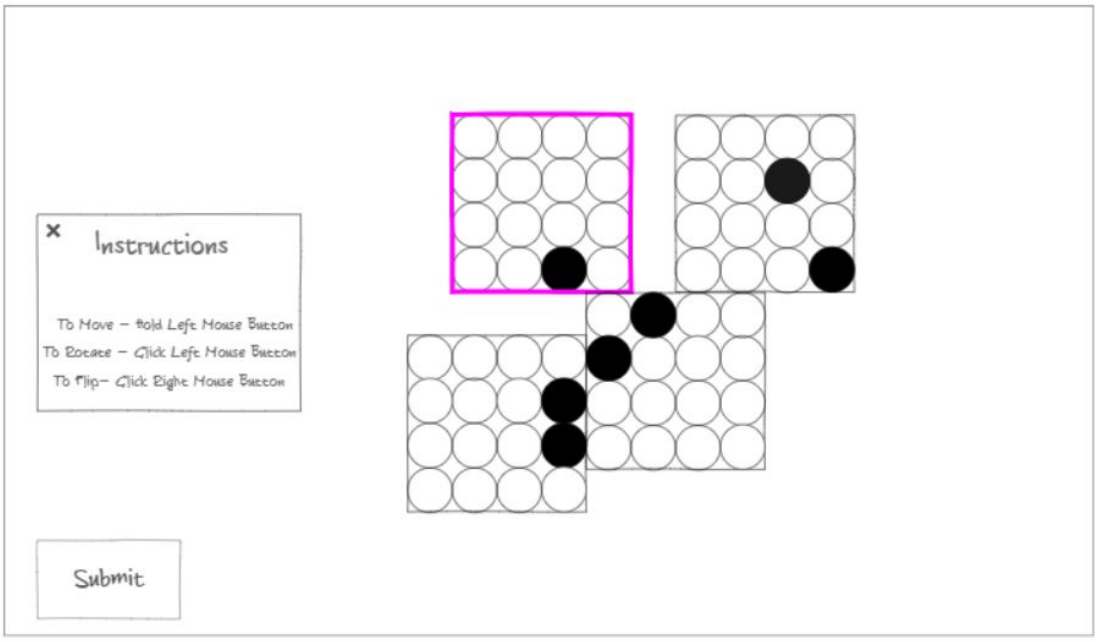

Figure 21: Compose Level Screen

7. Glossary

8. References