CS 319 - Object-Oriented Software Engineering
Analysis Report
Iteration 2

Q-Bitz

Group 1F

Burak Yaşar
Görkem Yılmaz
Hikmet Demir
Mert Duman
Mert Alp Taytak

Table of Contents

1. Introduction

We the members of the group 1F chose the game Q-bitz for the term project. This is an appropriate game to implement object-oriented programming principles. We will be implementing our game in Java since we thought that we can best express the object-oriented programming principles with Java and we are all comfortable with it. We are going to use Java Swing library since it will provide us to design User Interface (UI) and graphics. Our main goal by implementing this game is to get used to using the principles that we will learn in CS 319 course and to work with the object-oriented design concepts.

We can define the game as the following. Q-bitz is a magnificent visual agility game that will never lose its fascination. With 80 pattern cards and 16 cubes, players recreate the patterns as quickly as possible. From matching the card shown to having ten seconds to study a card and then remaking the pattern from memory, each variation of these fast-paced rounds requires a different set of visual and cerebral skills. But Q-bitz does not have to be played as a game. The cards and cubes can also be used as an exceptional thinking skill challenge for children or adults with short-term memory loss [1].

Q-bitz is a multiplayer game about building a pattern. The patterns are chosen randomly from a pile of cards and players try to match the pattern using cubes. Cubes are identical to each other but each face of a cube is different from its other faces. Depending on the specifics of the current round the player who builds the correct pattern fastest or the player who gets closest to the correct pattern wins the round. The player with the most rounds wins the game [1], [2].

Our aim in this project is to make a Q-bitz game that has the same rules as the tabletop version. However, we are going to focus on user experience. That’s why we will add four different modes which will be discussed in section 3.1 in detail to the original game which improve user experience and also increase the variety. Ultimately, our game will be

desirable for our target market, especially children like the game as it provides different modes and many functions.

Since the game will provide an online multiplayer mode, people can meet their friends through the agency of our game. People from different places can access our game and have fun with each other.

2. Overview

2.1 Menu

Main menu consists of several buttons. Functions of these buttons are explained in the following,

- Play: Leads to a game options menu. After desired options are selected the player can play the game in single player or multiplayer.
- Level Select: Leads to a campaign mode where player can select from pre-designed levels. For the levels, default game options and game mode are used. The player success is measured and given a rating upon level completion.
- How to Play: Opens the screen where game controls are shown and an explanation about how to play the game is written.
- Settings: Opens the screen where settings that are not directly related to the gameplay such as board size and game mode are displayed.
- Credits: Opens the screen where developers of this project are displayed.
- Exit: Closes the game.

2.2 Game Options

In the game options menu there are three settings,

- Board Size: The player can select to play the game in 3x3, 4x4 (default) or 5x5 cubes sized boards.
- Number of Players: The player can select to play the game with 1, 2, 3 or 4 players.
- Game Mode: The player can select from a list of game modes. These game modes are explained in more detail in functional requirements.

2.3 Settings

The list of settings are as follows,

- Window Options: The player can select window size of the game which means either he/she wants to play the game in full screen or default size
- Sound Options: The player can select the turn on/off the sound and turn on/off the background music.

2.4 Level Setup

The game is played in rounds. For each round there is a level. A level consists of players, cards, cubes, trays and boards. Following is the description for the previous level parts:

• Players: In each level there are 1 to 4 players. All of these are humans, there are no computer controlled players. Each player is given a board, a tray and cubes. If there are more than one players game will be online otherwise it is an offline game.

• Cards: The game is played by matching a pattern. The patterns are printed on cards. In each level, depending on the game mode, there can be many cards.

• Cubes: The players match the patterns on the cards using cubes, each face of a cube has a different pattern painted on it. Every player has a set of cubes. The amount of cubes depends on the board size settings. Also depending on the settings each cube in a set can be identical or have faces different from one another.

• Trays: Trays are where a set of cubes are stored before they are used. Each player has a tray.

• Boards: Boards are where players place the cubes to match the pattern given. Depending on the game options board size can be 3x3, 4x4 or 5x5 cubes. This will let the

more advanced players challenge themselves with the 5x5 board and let the beginners get used to the game playing in the 3x3 board.

2.5 Gameplay

Gameplay takes place between level start and level end. There will be multiple game modes implemented. Each game mode is different from each other gameplay and goals wise. However, the fundamentals do not change from game mode to game mode. There will be more information about game modes in the requirements section.

The fundamental gameplay consists of constraints, rounds, controls and goals. In each game mode there are different types of constraints. For example, in some modes patterns are visible for the whole round and in some modes they are hidden after a certain amount of time passes. The game is played in rounds and each round has one or more winners. After multiple rounds the player with the most points wins. Controls work as explained in the relevant functional requirements section. The goals also depend on the game mode. In some game modes finishing the pattern in time is enough but in other game modes the player has to be the first one to finish the pattern to win the round.

3. Functional Requirements

3.1 Game Modes

3.1.1 Pattern Matching [3], [4]

Mode one asks players to use their set of cubes to match the pattern shown on the card. The first one to complete the pattern wins the card and first pass is over. The player who wins the most card (who has the most matched pattern) wins the game.

3.1.2 Racing & Rolling [3], [4]

This mode implies online multiplayer game. Mode two asks the players to roll the cubes like dice and use as many of the cubes as possible to duplicate the pattern shown on the new card. Race to re-roll the cubes until the pattern is complete. First one done wins the card. (We have found it more exciting to re-roll each turn together instead of doing a free-for-all.)

3.1.3 From Memory in 10 Seconds [3], [4]

In mode three, players have 10 seconds to study a new card then remake the pattern from memory. This can be unbelievably difficult. The one who finishes first, or the one who has the most blocks in the correct position, wins the card. At the end of these modes, the player with the most cards is the winner.

3.1.4 Maximum Pattern Succeed in 2 Minutes

In fourth mode, player race against time which is given 2 minutes. Player should achieve the maximum number of patterns as best he/she can. In this mode, the player with the most cards is the winner as well.

3.1.5 Time Constraint Mode

Fifth mode states both players will have a time constraint on them and they will try to finish within this time limit. The players will earn points based on how much spare time they have left when they finish.

3.1.6 Painting Puzzle

The original game has 6 different shapes for each face of the cube. Beside that we will provide some original cubes as well. These cubes have different images or shapes on their faces and the player can use these faces to construct an image like a puzzle. For example, the players will try to complete a painting that has its pieces on each face of the cubes.

3.1.7 Different Cubes

In the last mode, different cubes will have different patterns as opposed to the original game, where each cube has the same patterns on each face. This will add the challenge of finding the correct cube on top of finding the correct face.

3.2 Controls

The game we are implementing is dexterity based in real life. Because of that, the controls are very important to give the user a satisfactory experience. Our implementation

uses mouse and keyboard. The mouse is used to pick up cubes from the tray and place them on the board. The keyboard is used to rotate the cubes to reveal other faces. To make it easier for the user, when a cube is picked up a small window will open to show all of the faces of that cube.

The mouse will also be used for camera controls. Spinning the mouse wheel will control zoom level, pressing right click and dragging will control camera rotation and pressing and dragging the third mouse button, the mouse wheel, will translate the camera.

Finally, the keyboard will have various shortcuts. Along with previously mentioned cube controls, there will be sound controls, shortcuts for switching between the views we implement and more controls as we need in the future.

4. Nonfunctional Requirements

4.1 Efficiency: Game Performance [5]

We are planning to add some animations into our game and we are going to make sure that these animations will not cause any performance issues in the gameplay. The standard FPS rate is usually 60 in most computers and we are planning to pass this limit or at least stay in that limit.

4.2 Usability [5]

A complex user interface could easily make a player feel uncomfortable and confused. Therefore we are planning to create a user friendly user interface which is easy to understand. There will not be lots of confusing buttons everywhere and the mechanics will be smooth and flexible.

In our game we are thinking of using high-quality images to create game pieces and for backgrounds. The animation tiles and the shadowing tiles that we are planning to use will make this game more than a simple board game in the eye of the player.

4.3 Maintainability [5]

Since most of the game objects have common attributes, they are open to reuse. This provides easier maintenance for the system. Moreover, when a common attribute need a change, it will be applied for only related class not every different object class. Since we will add source codes and prepare our reports other developers can keep updating the game with our permission.

4.4 Extendibility [5]

Our game will let some improvements, therefore it will be extendable for future upgrades. Additionally, new levels and scenarios can be added to system. Based on our

design game will be developed for the mobile applications however we will not interest in our game.

4.5 Availability [5]

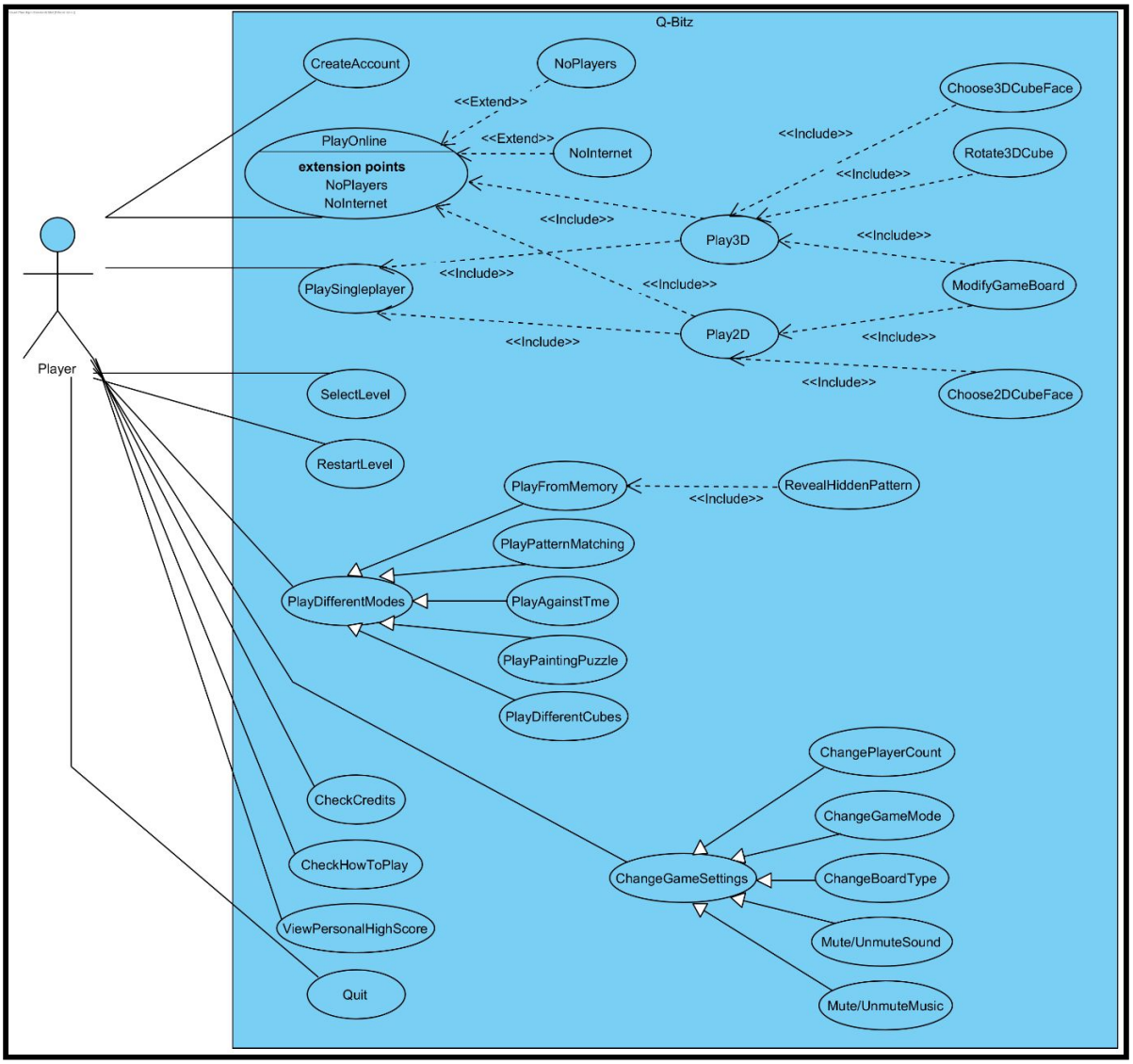
Since the Q-Bitz will be run by a jar file it will be platform independent game and it can be used all platforms who has JDK (Java Development Kit).

5. System Models

In this section, our game is modelled with functional model, dynamic models and object model respectively. Then, the corresponding mockup screens will be provided.

5.1 Functional Model

Here, the use case diagram of the Q-Bitz is given.



Textual Use Cases

Use Case Name:	PlaySingleplayer
Participating Actor:	Player
Entry Condition(s):	**“**Play” or “Select Level” button is pressed. Player count is chosen as 1.
Exit Condition(s):	Game is over. Exit button is pressed.
Flow of Events:	1. Player presses “Play” button. 2. Player chooses the game options. 3. Player presses “Start” button. 4. The game starts. 5. The game ends when the player finishes the pattern correctly.
Alternative Flow:	1. Player presses “Level Select” button. 2. Player chooses a level. 3. The game starts. 4. The game ends when the player finishes the level.

Use Case Name:	Play3D
Participating Actor:	Player
Entry Condition(s):	Player chooses the 3D option in game options.
Exit Condition(s):	Game is over. Exit button is pressed.
Flow of Events:	1. Player chooses the 3D option in game options. 2. Player presses “Start” button.

Use Case Name:	Rotate3DCube
Participating Actor:	Player
Entry Condition(s):	Player is in a 3D game. Mouse is held down over the 3D cube.
Exit Condition(s):	Mouse is no longer held down over the 3D cube.
Flow of Events:	1. When in a 3D game, player moves the mouse over the 3D cube and holds it down.

Use Case Name:	ModifyGameBoard
Participating Actor:	Player
Entry Condition(s):	Player is in a game. Player has selected a cube face.
Exit Condition(s):	Player modifies the game board or cancels face select.
Flow of Events:	1. Player presses “Play” button. 2. Player selects any game options. 3. Player starts the game. 4. Player hovers its mouse over any of the 6 cube faces (2D or 3D) and presses down. 5. While mouse is held down, players hovers the mouse over any of the slots on the board and releases the mouse.

Use Case Name:	PlayOnline
Participating Actor:	Player
Entry Condition(s):	Player count is anything but 1.
Exit Condition(s):	Game is over. Exit button is pressed.
Flow of Events:	1. Player presses “Play” button. 2. Player chooses the game options. 3. Player presses “Start” button. 4. The game starts. 5. The game ends when any player finishes the pattern correctly.

Use Case Name:	NoInternet
Participating Actor:	Player
Entry Condition(s):	Player starts the game with a player count 2, 3 or 4 and doesn’t have internet connection.
Exit Condition(s):	Player is alerted with having no connection.
Flow of Events:	1. Player presses “Play” button. 2. Player chooses the game options. 3. Player presses “Start” button.

	4. Player has no internet connection and is alerted that he/she has no connection. 5. Player is returned to game options menu with his/her chosen settings.
Use Case Name:	NoPlayers
Participating Actor:	Player
Entry Condition(s):	Player starts a multiplayer game but there are no other players to play with.
Exit Condition(s):	Player is alerted with there being no players.
Flow of Events:	1. Player presses “Play” button. 2. Player chooses the game options. 3. Player presses “Start” button. 4. If the player has waited over 3 minutes, he is alerted that there are no players to play with. 5. Player quits searching for other players.
Alternative Flow:	5a. Player keeps on searching for players although being alerted.

Use Case Name:	Choose2DCubeFace
Participating Actor:	Player
Entry Condition(s):	Player must be playing a 2D game.
Exit Condition(s):	Player selects a face.
Flow of Events:	1. Player presses “Play” button. 2. Player selects 2D from game options. 3. Player starts the game. 4. Player hovers its mouse over any of the 6 cube faces and presses down.
Alternative Flow:	1. Player presses “Select Level” button. 2. Player selects a 2D level. 3. Player hovers its mouse over any of the 6 cube faces and presses down.

Use Case Name:	PlayPatternMatching
Participating Actor:	Player
Entry Condition(s):	Player chooses “Pattern Matching” as the game mode.
Exit Condition(s):	Game is over.
Flow of Events:	1. Player chooses “Pattern Matching” as the game mode and starts the game.

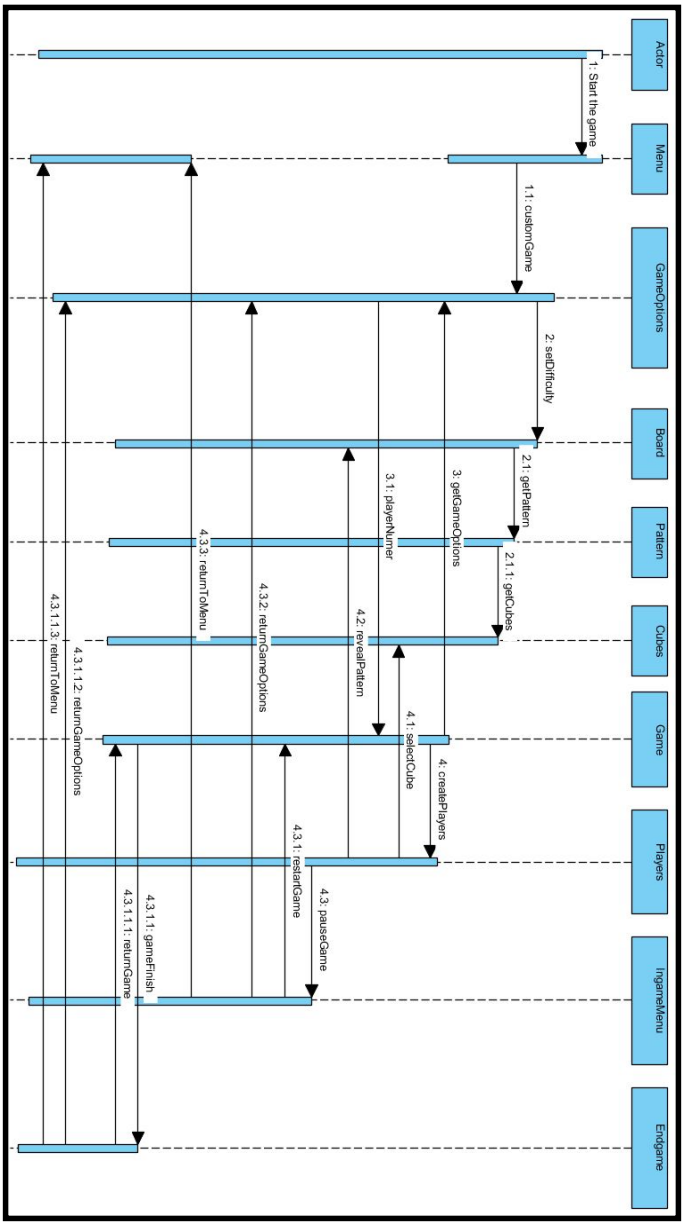
Use Case Name:	RevealHiddenPattern
Participating Actor:	Player
Entry Condition(s):	Player must be playing the “From Memory” game mode. Hidden pattern shouldn’t be revealed already.
Exit Condition(s):	The hidden pattern is revealed.
Flow of Events:	1. Player presses “Play” button. 2. Player selects the game mode as “From Memory”. 3. Player starts the game. 4. After giving up or a minute passes, player clicks the “Reveal Pattern” button.

Use Case Name:	ChangeBoardType
Participating Actor:	Player
Entry Condition(s):	Player is in game options menu.
Exit Condition(s):	Player exits the game options menu.
Flow of Events:	1. Player chooses the board type as 3x3, 4x4 or 5x5.

5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Custom Game Sequence

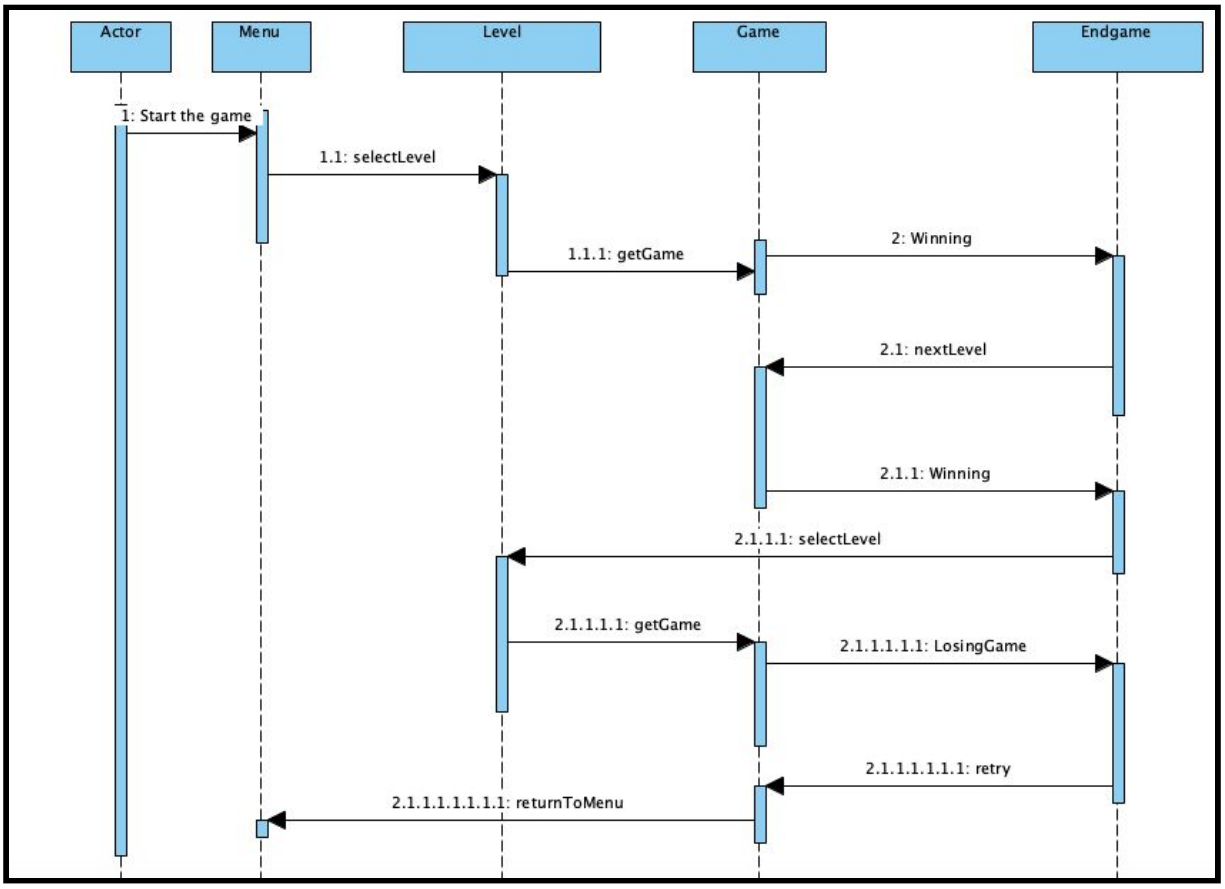


Scenario: User starts the game

Player enters the game and encounters with the main menu. When the player press the ‘Play’ button, game options menu pops up where player customize the game as he/she

wants. Depending on which difficulty he/she selects, the board is set up and the pattern is loaded according to board size where difficulty is a game option it specifies what will be the board size such as 3x3, 4x4, 5x5. Cubes are given to the user with the pattern as well then the game starts. After the game starts, the player can select one cube at a time and turn it in order to fit it into the pattern. Also, the player can pause the game and encounters with pause screen where he/she can turn back to the game options menu to change the difficulty of the game or turn back to the main menu. The player may restart the same game as well.

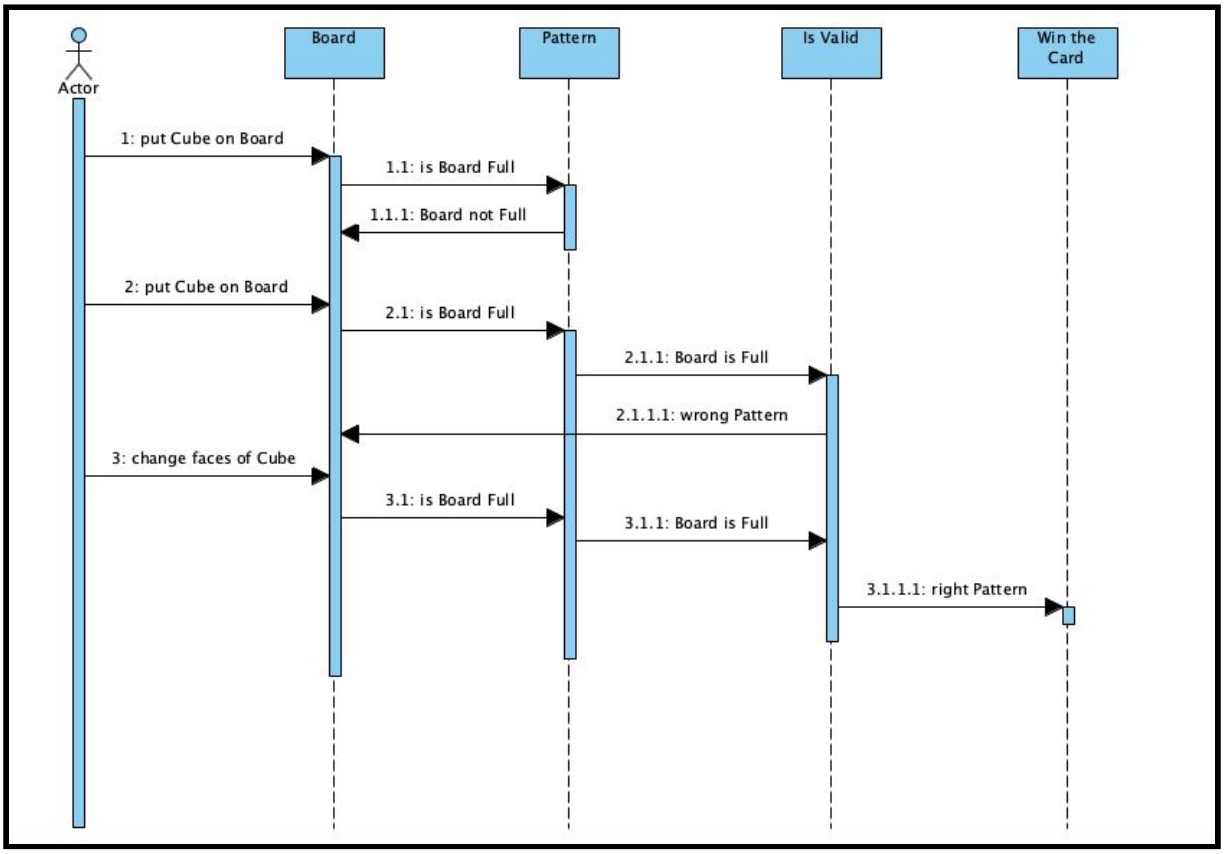
5.2.1.2 Winning and Losing Sequences of Time Constraint Mode



Player chooses select level option from main menu and select one level which is not locked from the levels and the game starts with default features (4x4 grid). In this case firstly player wins the game and the game is over, then player choose to go to next level then

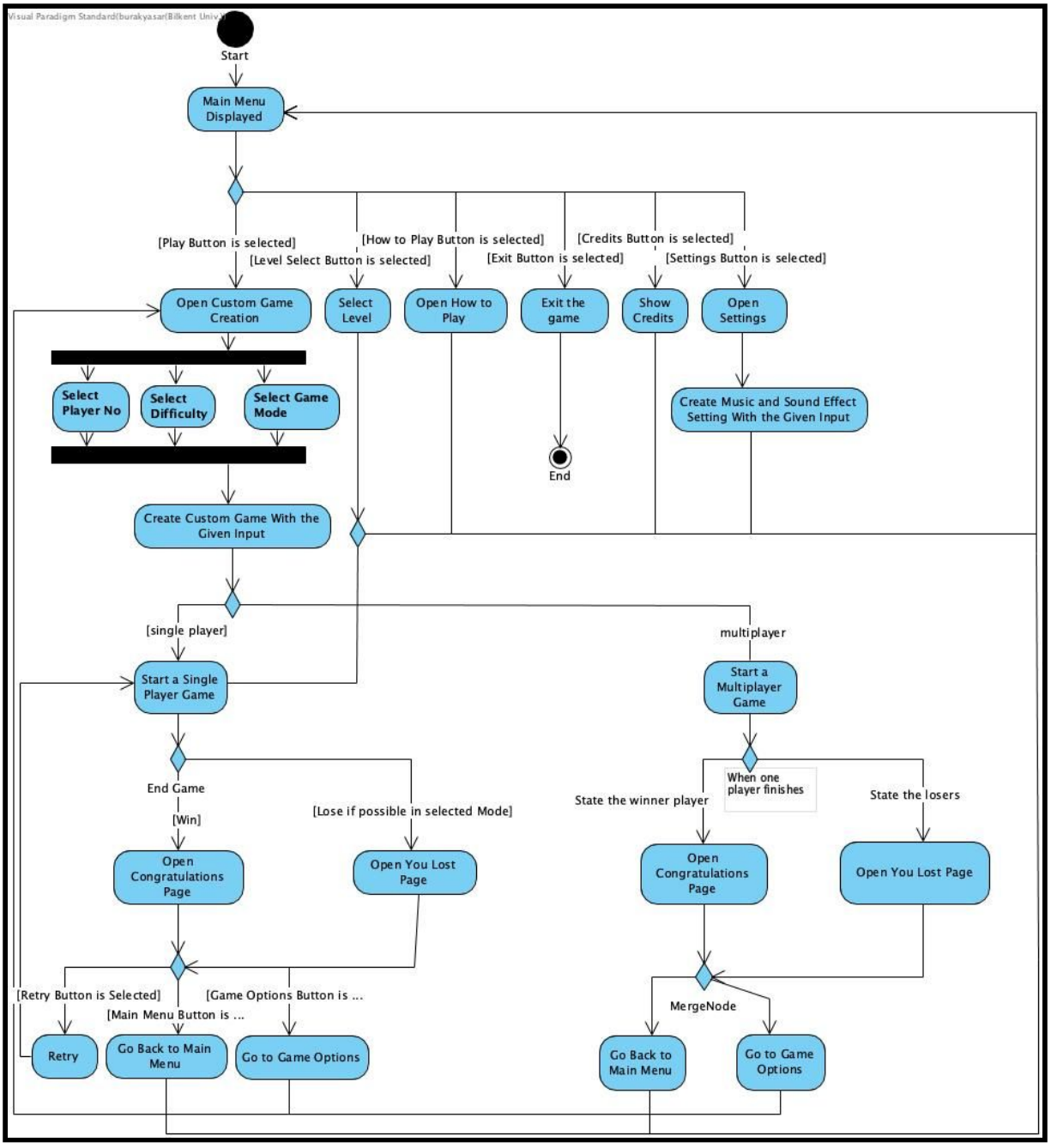
player wins again but this time player choose to go back to level menu and select a level. Within this game player loses and select to retry the game after that player go back to the main menu and sequence is over.

5.2.1.3 Solution Checking Sequence Diagram



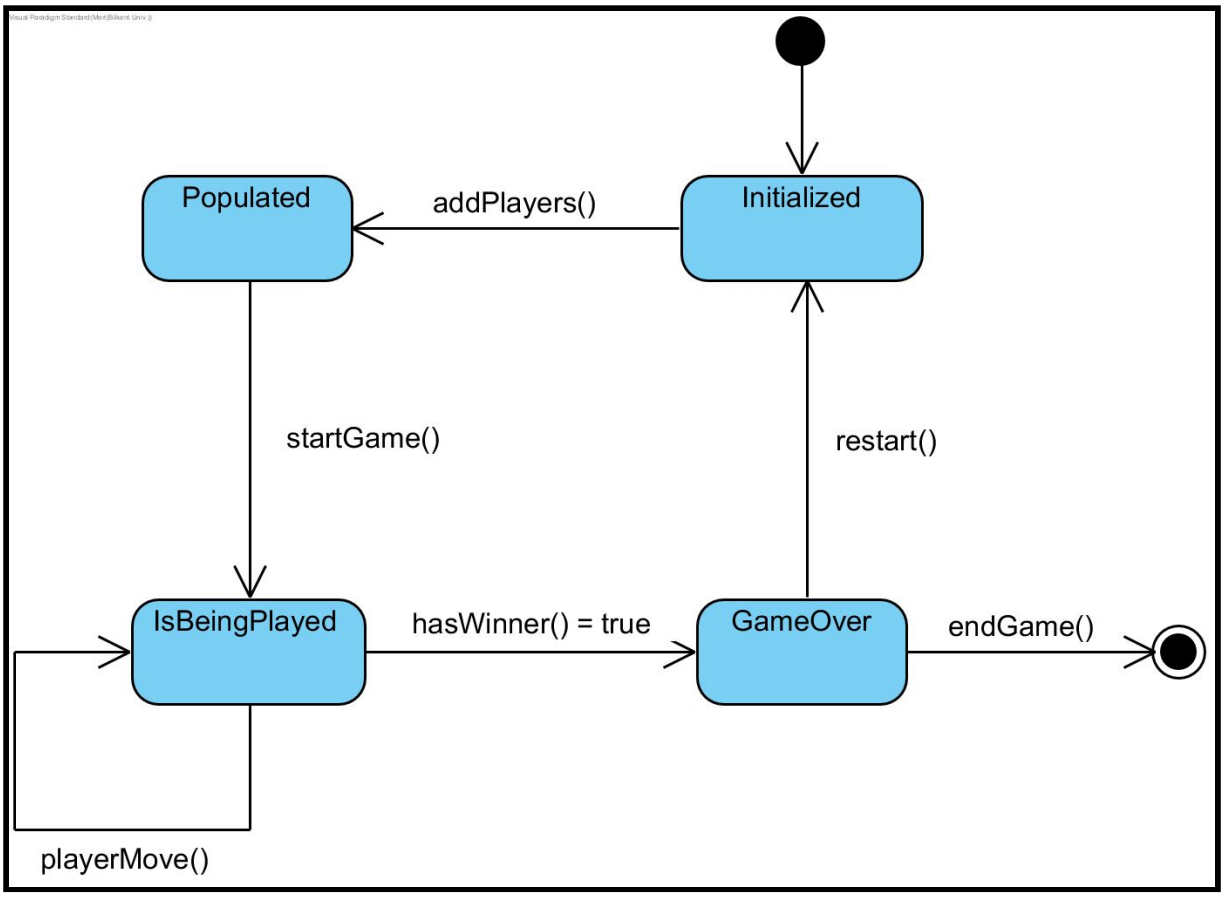
In this sequence player puts cube on the board and each time board will be checked if it is filled completely. If the board is not full, obviously player keep putting cubes on the board and when the board is full pattern will be checked whether if it is matched with the board which constructed by the player. If the pattern is wrong, it means game is not finished. Therefore, player has to change the faces of the cubes which are all on the board. When the board is matched with the pattern player wins the card which is the given pattern when the game starts.

5.2.2 Activity Diagram



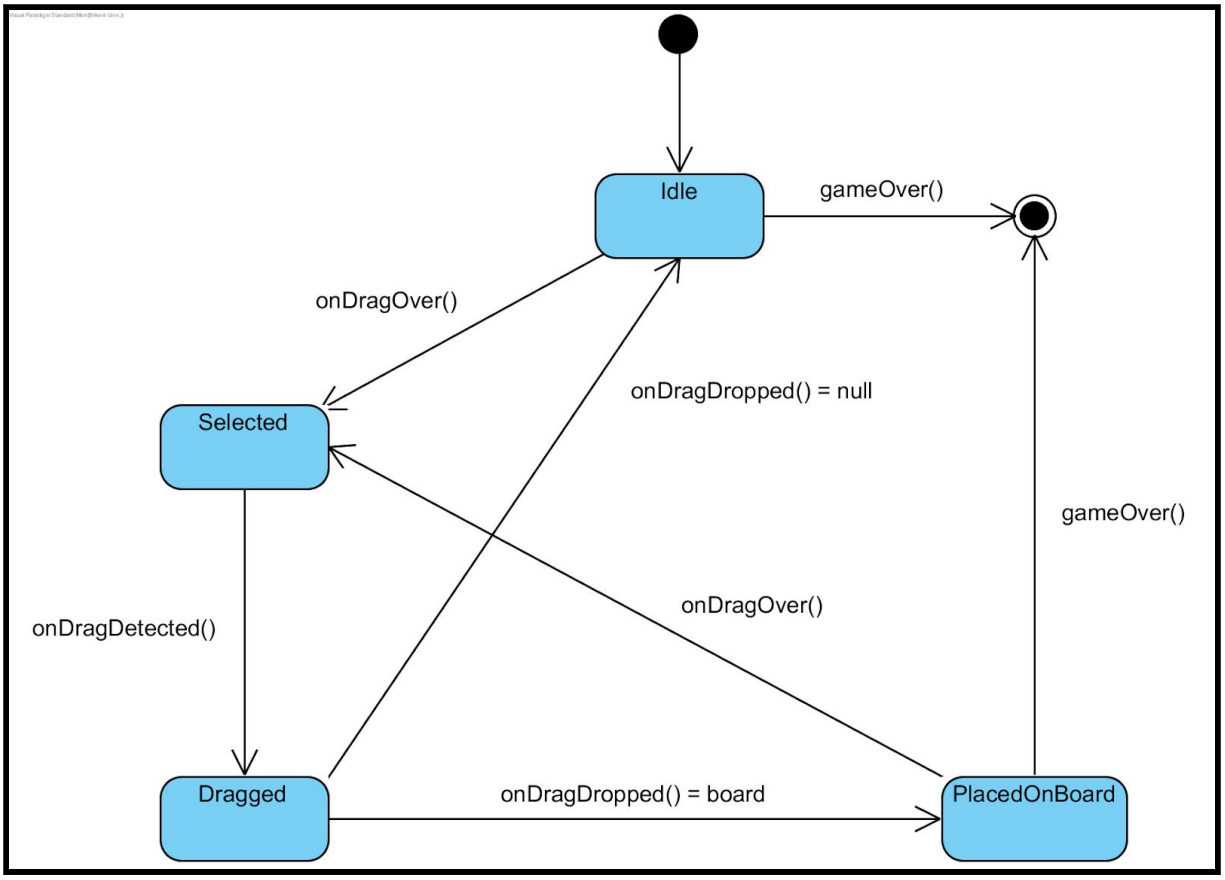
5.2.3 State Diagrams

5.2.3.1 Game State Diagram



Game state start with initialized state. After this state, game will go to Populated state with addPlayers function. When startGame function called game state will change to IsBeingPlayed state from Populated state. Game stays in IsBeingPlayed state until hasWinner function return true. playerMove function will call cause game state to be stay in IsBeingPlayed state. When hasWinner function returns true, game state will change to GameOver state from IsBeingPlayed state. In Game Over state, player can finish the game with endGame function or he/she can call the restart function and change state to Initialized state.

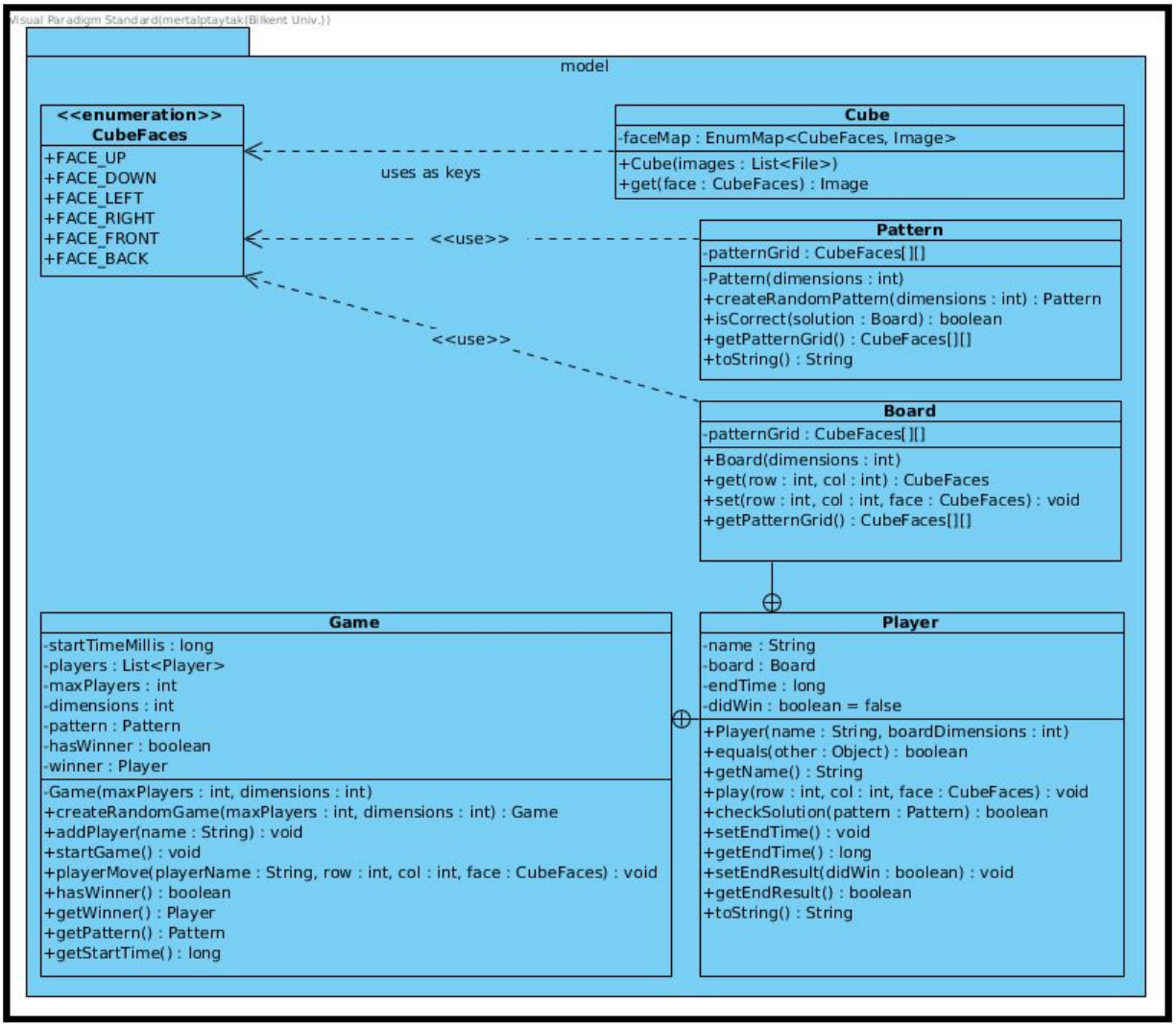
5.2.3.2 Cube State Diagram



Every Cube will start at Idle state. It will stay in Idle state until there is an action on Cube with onDragOver or gameOver action. Cube will change its state to Selected state from Idle state with onDragOver function. Cube state can be ended from Idle state with gameOver function. Cube can change its state to Dragged state from Selected state with onDragDetected function. In Dragged state, Cube can be transform to PlacedOnBoard state with onDragDropped function. If Cube onDragDropped method return null cube will change its state from Dragged to Idle. Cube will stay in PlacedOnBoard state until gameOver or it will change it's state to Selected State with onDragOver function.

5.3 Object and Class Model

5.3.1 Model Class Diagram



CubeFaces: An enumeration used to keep label the faces of a cube.

Cube: A class whose instances carry a map of cube faces to images. This mapping is used to render the faces of a cube instance.

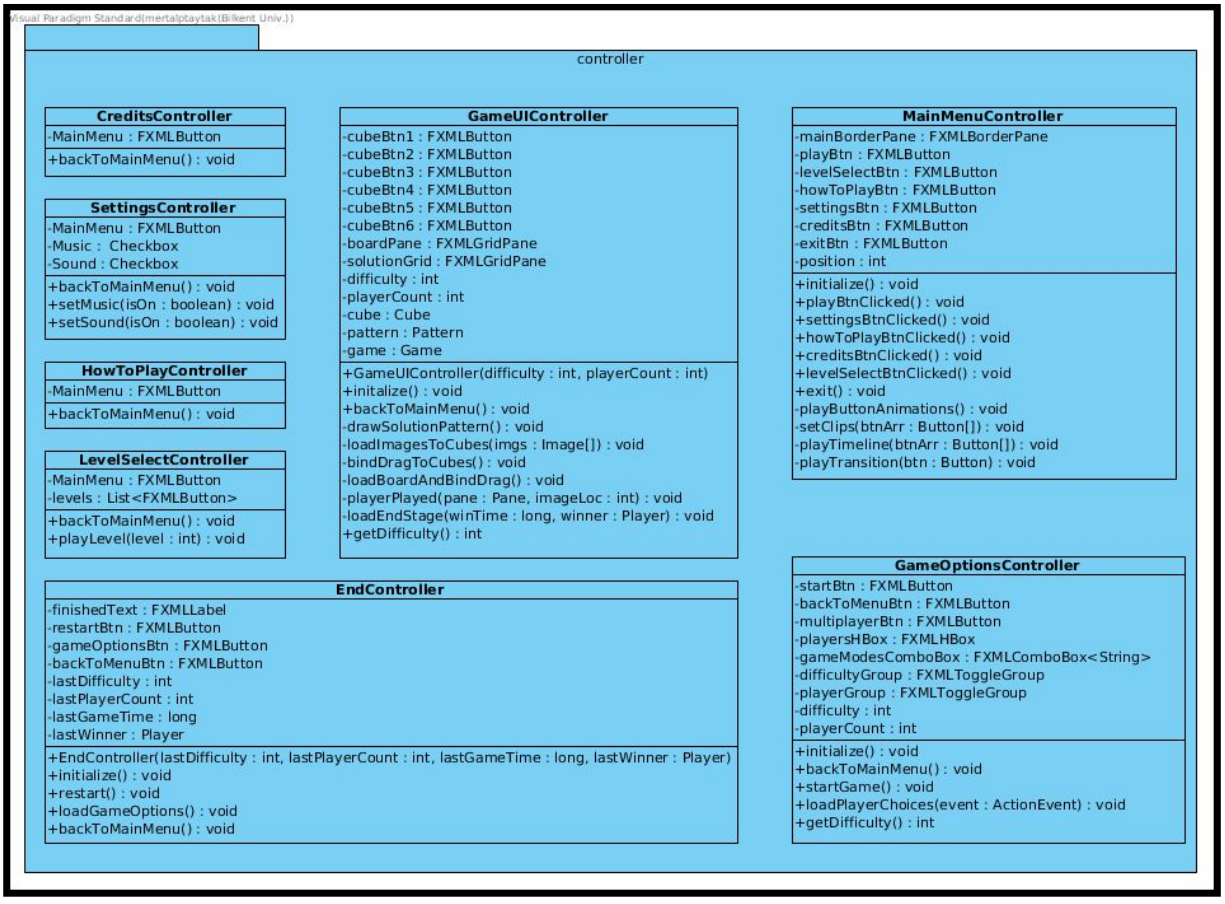
Pattern: A class that contains a initialized double array of cube faces. Any Board can be compared with the internal grid to check if they are equal or in other words if the Board is correct. Currently the patterns are created randomly, however there will be premade patterns in the next implementation.

Board: A class used to keep track of player moves. It contains a double array of cube faces. Each location on the board can be accessed through the use of set and get methods.

Player: A class used to represent the players that play the game. Along with the identifying information and the board, there are also other variables to keep track of the player state in a round of the game.

Game: A class used to represent a single round of the game. It contains players, the goal pattern and other information that is relevant to construct the game and keep track of the game status. The game is initialized through the use of pattern constructors and the player adder method. Then, the game is started and each move made by the players is recorded through the playerMove() method. After every move the board of the player is checked against the correct pattern to find if the player has finished the game and player time is tracked. Depending on the game mode, a winner is determined by different metrics.

5.3.2 Controller Class Diagram



MainMenuController: A class used to act as a bridge between different views. Some of the views can only be accessed through the main menu. Others eventually return to the main menu. MainMenuController is used to control the state changes and the main menu view itself.

HowToPlayController: A class used to display the tutorial and then, when the user is done, return to the main menu.

CreditsController: A class used to display the credits and then, when the user is done, return to the main menu.

SettingsController: A class used to display the settings and through the user interface elements set the sound and music options. Then, when the user is done, returns to the main menu.

LevelSelectController: A class used to allow the player to play in puzzle mode. In the puzzle mode, player plays through premade levels and receives a performance rating for each level. This controller is used to select the level to play or return to the main menu.

GameOptionsController: A class used to display and choose from options such as player count and difficulty for a round of the game. Then, either the player starts the game or returns to the main menu.

GameUIController: A class used during a round of the game to communicate with the view and the model. It displays the current model and updates the model according to the user input from the view. Then, when the round is over it switches the control to another class.

EndController: A class used to display end of the game information to the user and allow the user to replay the game or return to the main menu.

5.4 User Interface

In this section the screen mockups of our game will be given.

5.4.1 Main Menu

Figure 1

5.4.2 Game Options

Figure 2

5.4.3 Game Screen

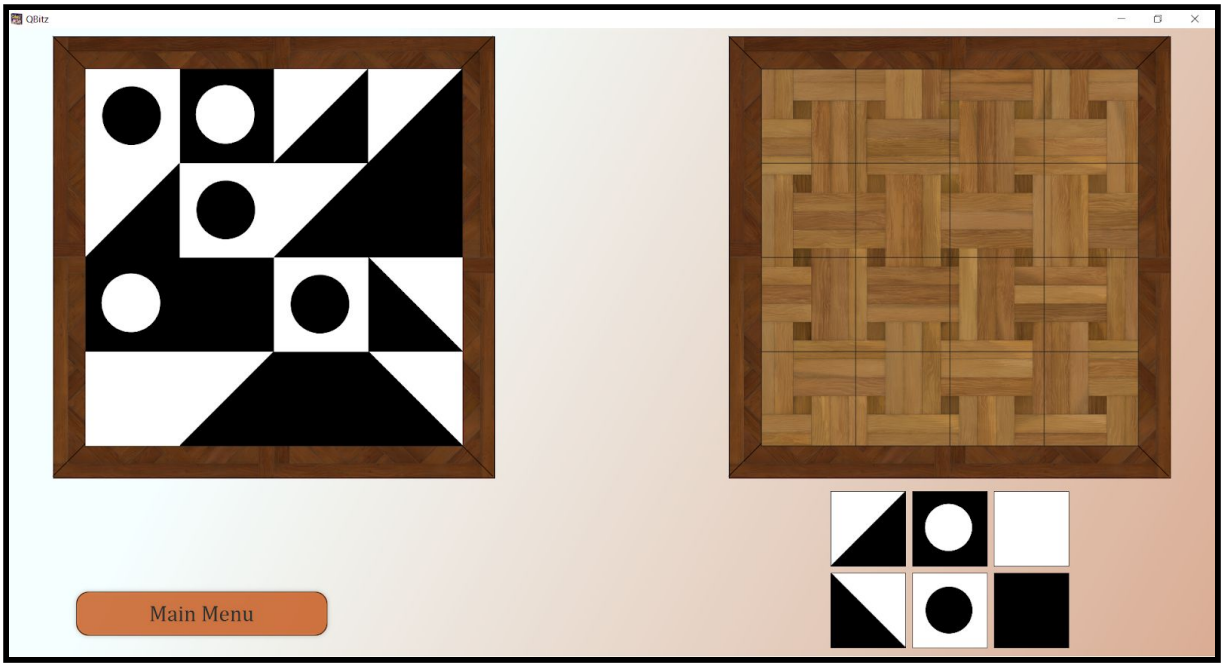


Figure 3

5.4.4 Level Select

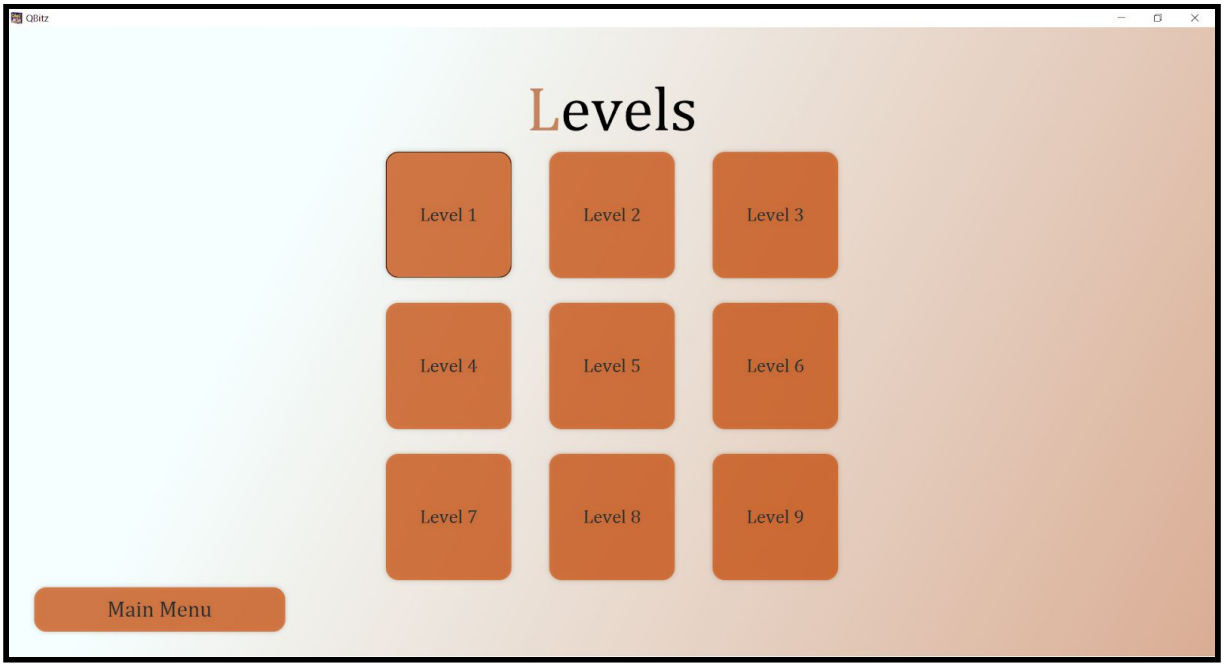


Figure 4

5.4.5 How To Play

Figure 5

5.4.6 Settings

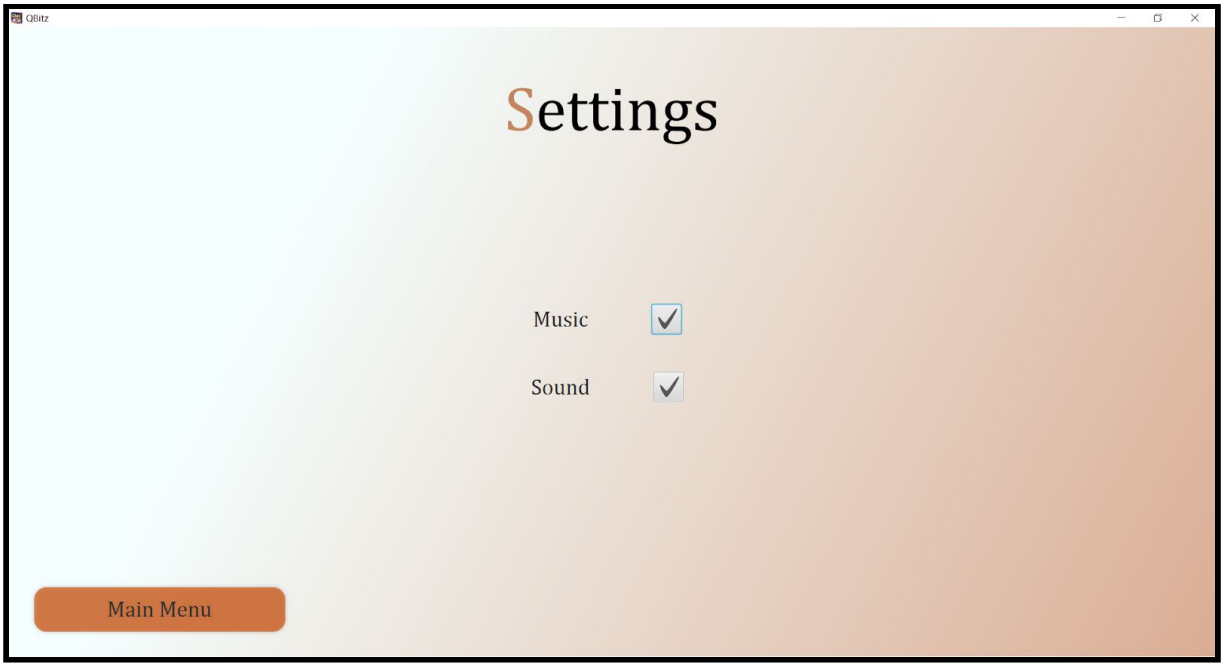


Figure 6

5.4.7 Credits

Figure 7

5.4.8 End Game Screen

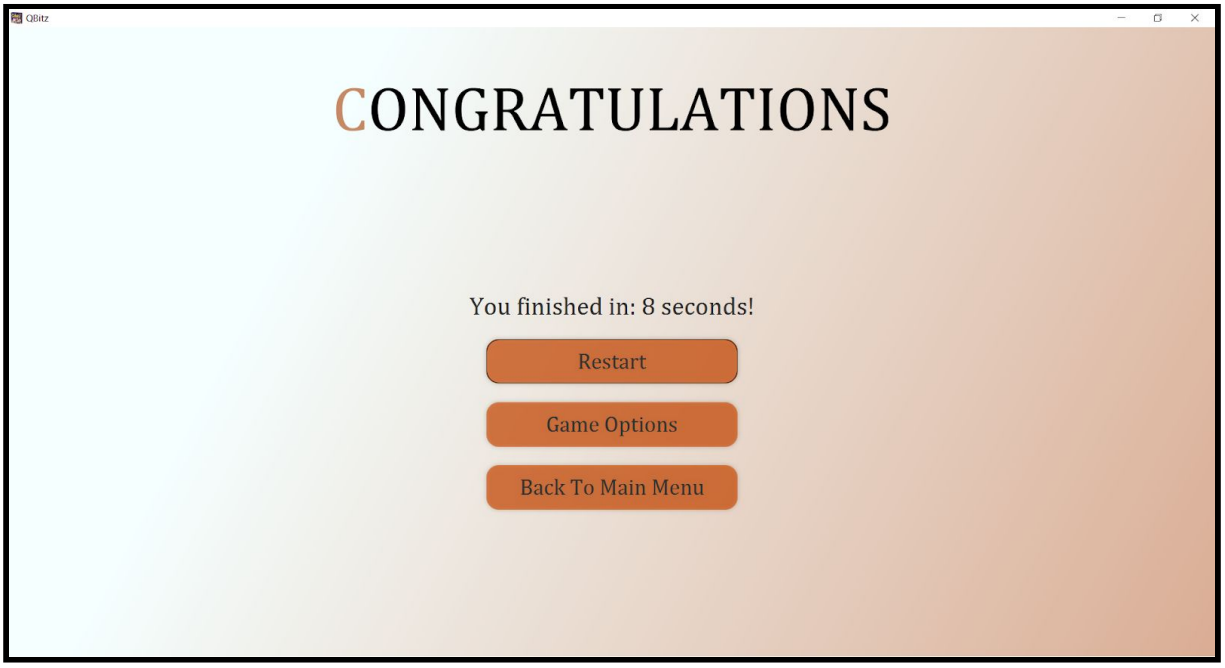


Figure 8

6. Improvement Summary

We added extra use cases and classes to our design and clarified the existing features after we began to implementation and those changes can be seen in our system models. We fixed our report as stated in the feedback. Changes are listed in the following:

- In the last paragraph of the Introduction we clarified what are our contributions to the game.
- We removed the unnecessary statement in section 2.2. Also, we stated what the window size means in section 2.3.
- In which cases the game will be online is specified in section 2.4.
- In section 3.1.1 and section 3.1.6, corresponding game modes have been explained in detail.
- Non functional requirements types are corrected and explained how they related. We merged the tile options and non-complex user experience within a one section called usability since both serves to the better usage of the game. Also, we add three more non functional requirements which are maintainability, extendibility and availability.
- In activity diagram notation is fixed to show that we can only exit, if we click exit menu in main menu.
- Use case diagram is updated and it is also written in textual form to explain better.
- Class diagram is given with the controller this time.
- Sequence diagrams are renewed and redundant parts are associated with the corresponding sequence diagrams.
- References are numerated and each of them is used in the report where the necessary parts.

7. Glossary and References

[1] “Q-Bitz Jr.” TOYTAG, https://www.toytag.com/products/q-bitz-jr.

[2] TimeToPlayMag. “Q-Bitz from MindWare.” YouTube, YouTube, 3 Sept. 2013, https://www.youtube.com/watch?v=j4PEAbkT780.

[3] “Q-Bitz.” Timberdoodle Co, https://timberdoodle.com/products/q-bitz.

[4] “Q-Bitz.” Amazon.co.uk: Toys & Games, 7 May 2010, https://www.amazon.co.uk/Green-Board-Games-44002-Q-Bitz/dp/B0031P91LK.

[5] Bruegge, B., & Dutoit, A. H. (2014). Object-oriented software engineering: using UML, patterns, and Java. Harlow, Essex: Pearson