Object-Oriented Software Engineering

CS319

Term Project
Analysis Report

Q-Bitz

Group No: 3F
Group Name: Wasted Potentials

Mertkan Akkus	21602951
Yasin Alptekin Ay	21601849
Ahmet Furkan Biyik	21501084
Ramazan Mert Cinar	21601985
Yaman Yagiz Tasbag	21601639

Table of Contents

Introduction

Q-Bitz is a board puzzle game composed of three rounds.Round one of this games all about speed as players race to manipulate their cubes to copy the pattern on the Q-bitz pattern card. In round two, players roll all of their cubes on the table like dice. Using the cubes as rolled (face up), players place as many cubes as possible in the tray to recreate the pattern shown on the card. Players race to re-roll all remaining unusable cubes until they roll the shapes that are needed to complete the pattern. Round three really tests brain power: players study the pattern card for ten seconds, flip it face down and then re-create the pattern from memory![1]

We started to plan by deciding the most feasible platform for the game and decided to implement a desktop application since the game is more convenient to play in bigger screens. It is more comfortable to play the game using mouse and keyboard than playing on touch screen.

The original game is made mainly for children, since the base features of the game are not complicated, and it is easy for an adult to solve the puzzles rapidly. We decided to expand the target audience by modifying game so that it is more sophisticated and charming. To make the game more attractive, we decided to develop more on adding new game modes. By adding more game modes, we think that we address not only children but also adults, as the game modes require the player to be strategic and to be good at time management. The additional game modes can be listed as follows:

- Switch Mode
- Survival Mode
- Daily Challenge

The game modes will be explained further in section 2.

What also make the game appealing are visual and auditory effects and themes that we designed. We are also planning to produce game music and effect sounds ourselves. The player will be allowed to adjust the sound level of the music as well as effect sounds and master sound.

We will explain more about the features that we have designed in the upcoming sections but the whole list of added features can be seen below:

- Game Music
- Effect Sounds
- Themes
- Singleplayer Mode
    - Daily Challenge
    - Survival Mode
- Multiplayer Mode
    - Switch Mode
    - Classic Mode
- Tutorials
- Custom Made Environment (see interfaces)

Overview

Components

Board

The board (see Figure 1-1) is the canvas where the cubes will be placed. By default, the board will be loaded as 4x4 but will be

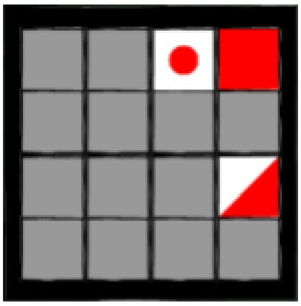


Figure 1.1 The board

adjustable by the player. The cubes can be placed, rotated and displaced in any order decided by the player.

Cubes

The base component of the game is cubes (see Figure 1-2). There are 6 faces but 5 distinct faces in the cubes. All cubes are identical in the game. The cubes will be placed in the game board by the player. All the players have the same amount of cubes in a game and it's equal to the number of slots in the board.

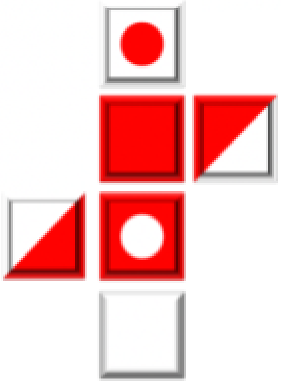


Figure 1.2 Cube Faces

Game Cards

The game cards (see Figure 1-3) are the mission of the players. They will always be displayed in the game and maximized when "tab" key is pressed.

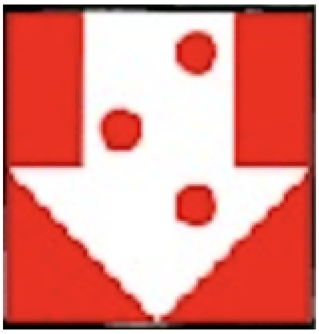


Figure 1.3 Sample Game Card

How to Play

There are various rules for each mode and they will be explained in the related sections, but the base rules and game-flow are as follows:

- At the beginning of the game, a game card is assigned to a player.
- The objective is to have the same shape as in the game card, on the board.
- The player will move, rotate and arrange the cubes as he/she wishes.
- When the objective is completed, the game will move on according to the game mode.

Game Modes

The game can be played both as a singleplayer and also multiplayer. Both have two more sub-modes.

Singleplayer Mode

There are two sub-modes of the singleplayer mode: Daily Challenge and Survival Mode.

Daily Challenge

In the Daily Challenge Mode, a player is assigned predetermined game cards and those game cards will change daily. When the player solves a game card, the next one will appear. The player will be placed in the daily leaderboard according to the time elapsed to solve all the game cards. The size game board will be 8x8 for the Daily Challenge Mode and of course, will not be adjustable.

Survival Mode

In the Survival Mode, a player has a predetermined time limit and will be randomly assigned game cards until the time is over. The time will be incremented by certain number of seconds – which is not determined yet – when each successful submission is made. The score will be the number of cards solved at the end of the time.

Multiplayer Mode

There are two sub-modes of the multiplayer mode: Switch Mode and Casual Mode.

Switch Mode

In the switch mode, there will be 2 players. At the beginning of the game both will be assigned a game card. In every 15 seconds, the boards will be switched, and the players will be progressing on the other board. The winner will be the one who finishes first, no matter which board he/she completed.

This mode is one of the most challenging ones since the strategy is crucially important. A player might choose either to complete the board or to disarrange the opponent’s board.

Classic Mode

In the Classic Mode there may be up to 4 players. Each of them is assigned the same game card and the first one to finish wins the round. This mode was already in the base game.

Functional Requirements

Additional Functional Requirements

Spectator Mode

This mode allows users to watch the games. Only multiplayer game modes can have spectators. A user can join a lobby as a spectator if he/she selects the lobby from the list and clicks the “Spectate” button (see Figure 7.10). Spectators can join the lobby even if the lobby is full or playing. In spectator mode, game boards of all the players in the lobby are displayed to spectators with the players’ names on top of their boards. Each time a player makes a change on his/her board, the spectator’s screen is also updated. When a player finishes the game, winner’s name is displayed to the spectators.

Key Bindings

A player can bind the cube rotation functions to any key on his/her personal preference. Key binding settings can be accessed by pressing the “Key Bindings” button on the settings screen (see Figure 7.20).

Game Modes

Classic game mode

This is a multiplayer game mode. Players will join or create lobbies to play this mode. Up to 4 players can participate in a lobby to play. Players will compete against each other to complete more game cards until a fixed amount of game cards have been completed. After each puzzle is completed, the player who completes first will gain the card. The player with the highest amount of game cards wins the game.

Daily challenge

This is a single player game mode. Each day there will be a daily challenge assigned by developers, where each player is going to try to complete the challenge in the shortest time. Their completion times will be displayed in the leaderboard.

Switch mode

This is a multiplayer game mode. Players will join or create lobbies to play this mode. 2 players will compete against each other.2 boards will be switched between two players; each player will interact with the board for a fixed time interval. The player who completes the puzzle wins. Players can disarrange the cubes on the board so players will use various strategies when playing this mode.

Survival mode

Survival mode is a single player game mode in which player tries to solve as many puzzles as they can in a time interval. Player will start with a specified time, each time player solves a puzzle, he will be given bonus time. So in this mode players will challenge themselves against time.

Lobby System

A lobby system will be implemented for multiplayer game modes. A player can create or join a lobby. In the lobby creation screen, the creator of lobby will specify the name, game mode and player limit of the lobby. There will be an option to make the lobby private, so players can only join with an invite. In the lobby list screen players will be able to sort or filter the lobbies to find lobbies with desired features. There will be a chat box in the lobby so players in that lobby can communicate with each other. Lobby admin will be able to start the game. The lobby admin will be able to invite players to the lobby or kick players from the lobby.

The lobby admin needs to enter the nickname of the player to invite him/her to the lobby. The invited player will receive a pop-up screen with the nickname of host asking if they want to join the lobby or not. If the invited player accepts the invitation, he/she will be moved to the lobby, if the invitation is rejected, player will remain in the main menu.

How to play

Players will access to this screen via the main menu. In this screen, game rules and tutorials are given for each game mode. Each game mode is explained under a tab. The tutorial is supported with pictures so that player can understand the basics of the game easier. The tutorial screen supports guides for classic, switch, daily challenge and survival game modes of the game. Game controls are also explained with pictures in this screen.

Settings

This screen is accessible from the main menu, here, player can change basic settings of the game. Player can submit a new nickname. Player can change the resolution and make the game full-screen. Sound settings can also be changed here. Player can adjust master volume, effects volume and music volume.

Credits

This is the screen with the information of developers and the link to GitHub page of the project. Players can access to this screen from main menu.

Non-Functional Requirements

Additional Non-Functional Requirements

Server Response Time

The response time of the server should be less than 1 second to provide a better environment to the player since in some circumstances, the user interface is rendered depending on the server response. For instance, if a user joins a lobby as a spectator, the screen should be updated after every player’s move.

Theme Music Transition

Musics should be changing smoothly, ie. in 500 milliseconds, after one is finished in order not to bother the player and not to give the impression of a possible error that might have been occurred in in-game media player.

User-Friendly UI and Performance

Since the time plays a very significant role in this game because players will compete with each other and against time, the performance is very important and while they are struggling with these challenges it is really desired to have a smooth game play experience and this is all about UI.

- In the software version of the game, the navigation should be handled with menus which are simple and understandable.

- Board game is designed as a mind game for children, so even a child of a small age should be able to navigate through the menus.
- Joining a multiplayer game should not be overwhelming. Players should be able to play with their friends by filtering the lobbies by name and finding the lobby their friends are in or just by accepting an invitation which is sent by their friends - a screen will pop up when a player is invited to a lobby-.
- In the lobbies screen, finding a lobby should be easy. Player should be able to sort the lobbies or filter them by particular properties.
- The software should be responsive. Selecting, rotating and placing a cube should feel smooth to the player.
- Response time of the UI should be less than 15 milliseconds.
- Time for a new player to get used to the game should be less than 10 minutes.

Ease of Control

Q-Bitz is a time-based game, which means it requires players to play as fast as they can to win the game. Considering this feature of the game, its desktop version must be easy to play. Requirements for such a convenience can be listed as following:

- Cube selection should not be complicated. In the original board game selecting a cube from a pile of cubes is not difficult. However, in the software such a selection can be complicated. This complication can be fixed by showing the cubes to the player one by one. Next cube will be shown to player after placing the previous one.
- Cubes must be easy to rotate. A player should not spend all his/her time on rotating the cube. Player should be able to rotate the cube using arrow keys or with a click of a rotation button.

- In case of a misplacement, a player should not hassle with displacing the cube, instead, the player should be able to select the cube on the game board and just rotate it.
- Mistakes in control should not occur more than 0.2 moves per control move.

Additional Features and Attractiveness

The original board game only consists of one game mode with three steps which can make unappealing to adults. The base game is also simple and only requires players to memorize and not use any strategies. The game can be made more interesting and its target audience can be extended by adding some features that can be listed as following:

- There should be additional game modes in which players can use various strategies against their opponents and use the time more efficiently.
- There should be a leaderboard, so players can challenge with others and try to score highest which will make the game more competitive.
- Game music and sound effects should be implemented so that players will have a better experience when interacting with a cube or the board and they will enjoy the game more when listening to game music.
- In the software version of the game, a chat system in the lobby should be implemented so that players can communicate with each other.
- 7 of 10 players should find the gamemodes fun and engaging.

System Models

Use-Case Diagram

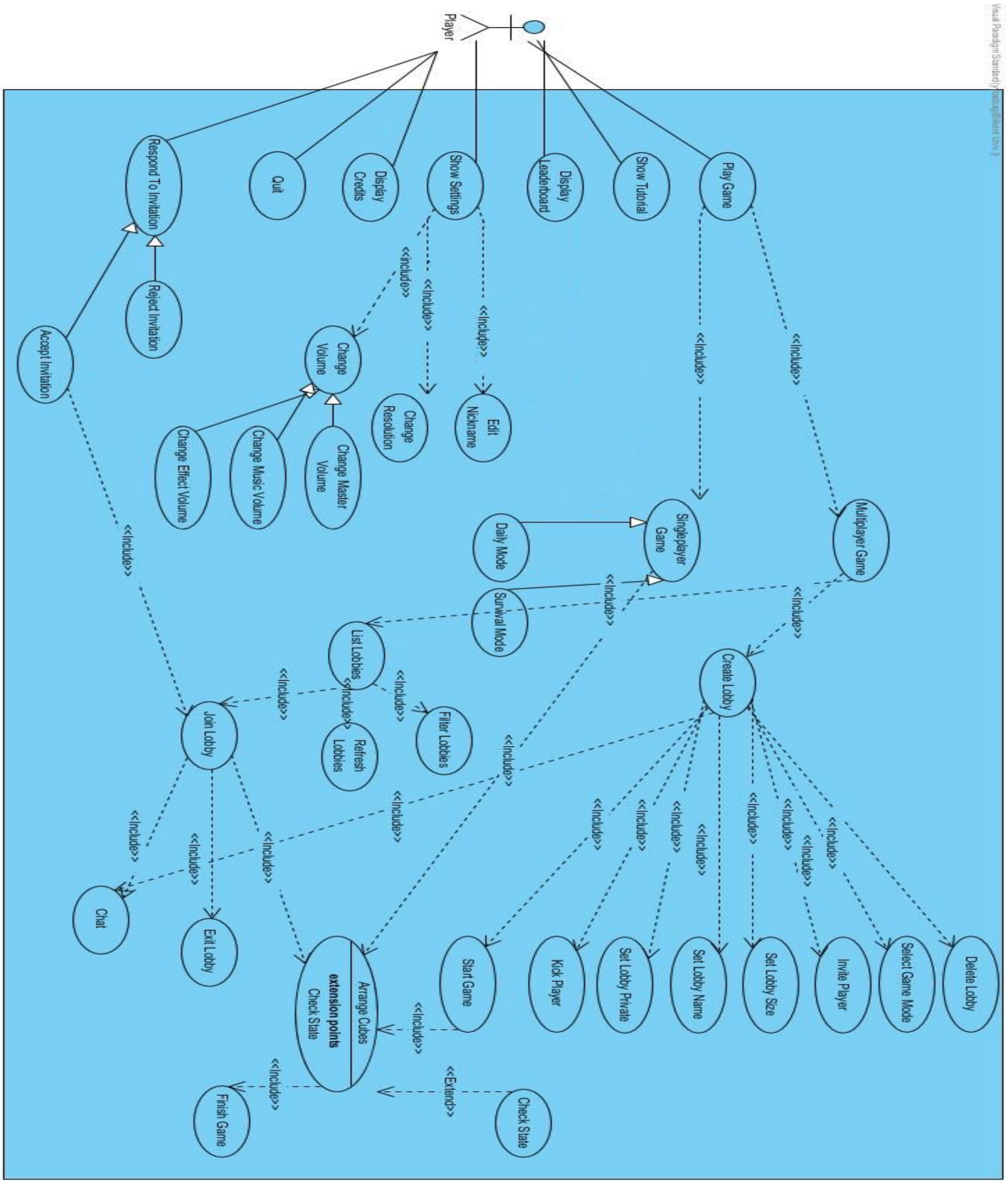


Figure 5.1.1 Use Case Diagram

Use Case #1

1. Use Case: Play Game
1. Participating Actors: Player
2. Entry Condition: Player chooses play game
3. Exit Condition: Player chooses Go Back

Scenario #1

1. Player chooses multiplayer

Scenario #1.1

i. Player is in a classic game mode lobby
ii. Game starts by lobby master
iii. Player arranges the cubes according to a game card
iv. Game ends and displays scoreboard

Scenario #1.2

i. Player is in a switch game mode lobby
ii. Game starts by lobby master
iii. Players arrange the cubes according to game card
iv. Boards switch when time is up
v. Players arrange or disarrange the cubes
vi. The last two previous processes repeat until one player completes his/her board correctly.
vii. Game ends and displays winner

Scenario #2

1. Player chooses singleplayer

Scenario #2.1

i. Player chooses survival game mode

ii. Player arranges the cubes according to a game card

iii. Player gains bonus time if he/she completes a game card correctly

iv. Game ends when the time is up

v. Game displays score

Scenario #2.2

i. Player chooses daily challenge game mode

ii. Player arranges the cubes according to a game card

iii. Game ends when player completes all challenges

iv. Game displays leaderboard

Use Case #2

1. Use Case: Tutorial
2. Participating Actors: Player
3. Entry Condition: Player chooses tutorial
4. Exit Condition: Player chooses Go Back

Scenario #1

1. Player chooses the game mode for the tutorial of the desired mode
2. Player goes to next page

Use Case #3

1. Use Case: Leaderboard
2. Participating Actors: Player
3. Entry Condition: Player chooses Leaderboard

4. Exit Condition: Player chooses Go Back

Scenario #1

1. Player chooses Leaderboard for the desired game mode

Use Case #4

1. Use Case: Settings
2. Participating Actors: Player
3. Entry Condition: Player chooses Settings
4. Exit Condition: Player chooses Go Back

Scenario #1

1. Player chooses Edit Nickname
2. Player clicks submit

Scenario #2

1. Player chooses change resolution
2. Player clicks on OK

Scenario #3

1. Player chooses change volume

Scenario #3.1

i. Player changes master volume

Scenario #3.2

i. Player changes effect volume

Scenario #3.3

i. Player changes music volume

Use Case #5

1. Use Case: Credits
2. Participating Actors: Player
3. Entry Condition: Player chooses Credits
4. Exit Condition: Player chooses Go Back

Scenario #1

1. Player chooses Credits
2. Names of the developers are displayed

Use Case #6

1. Use Case: Accept invitation
2. Participating Actors: Player
3. Entry Condition: Player receives game invitation
4. Exit Condition: Player accepts or rejects invitation

Scenario #1

1. Player Accepts invitation
2. Player joins to Lobby

Scenario #2

1. Player Rejects invitation
2. Player remains in main menu

Dynamic Models

Activity Diagram

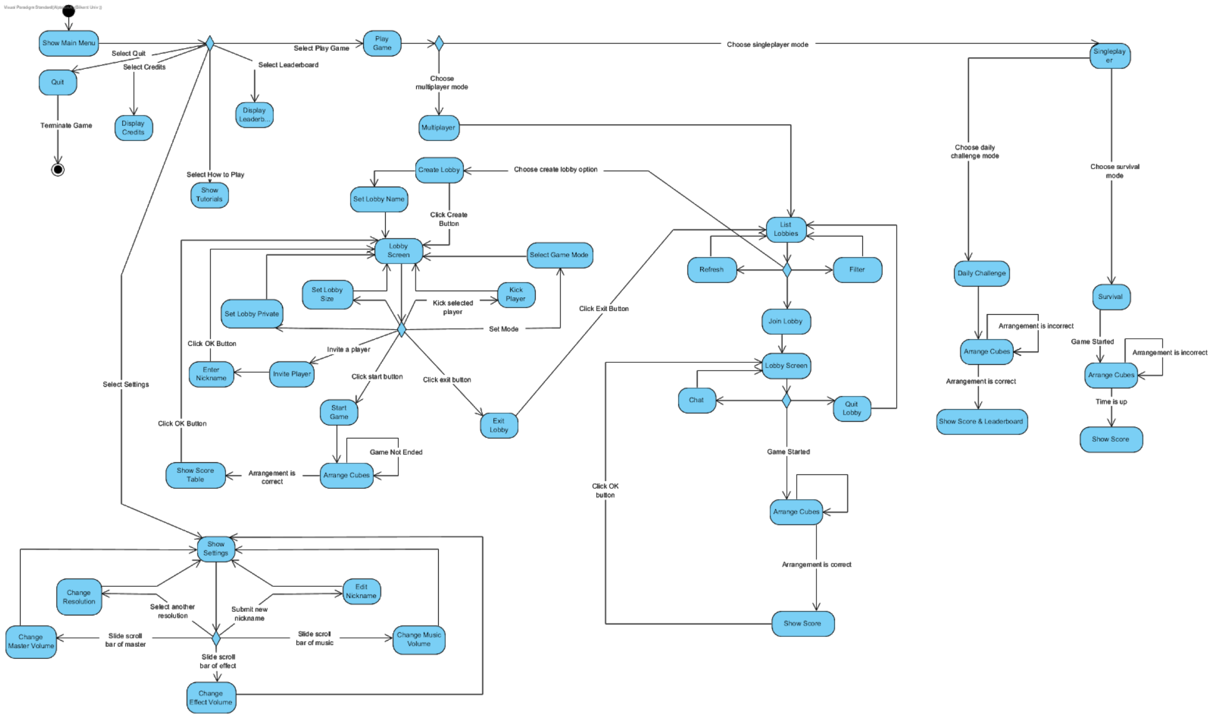


Figure 5.2.1 Activity Diagram

The activity diagram of the game is shown in Figure 5-2-1. The diagram starts from the very beginning of the game (main menu) and demonstrates all possible flows. There are 6 use cases of the game as explained in section 5.1: Play Game, Tutorials, Leaderboard, Settings, Credits, Quit. Those are all considered in activity diagram with their sub-flows.

State Diagrams

The state diagrams that belong to Kubitz are designed for the game modes, and for all the game modes, there is one nested state which is called Cube Management. This Cube Management State is describes the actions made on a single Cube shown in figure 5.3.1

Cube Management State Diagram

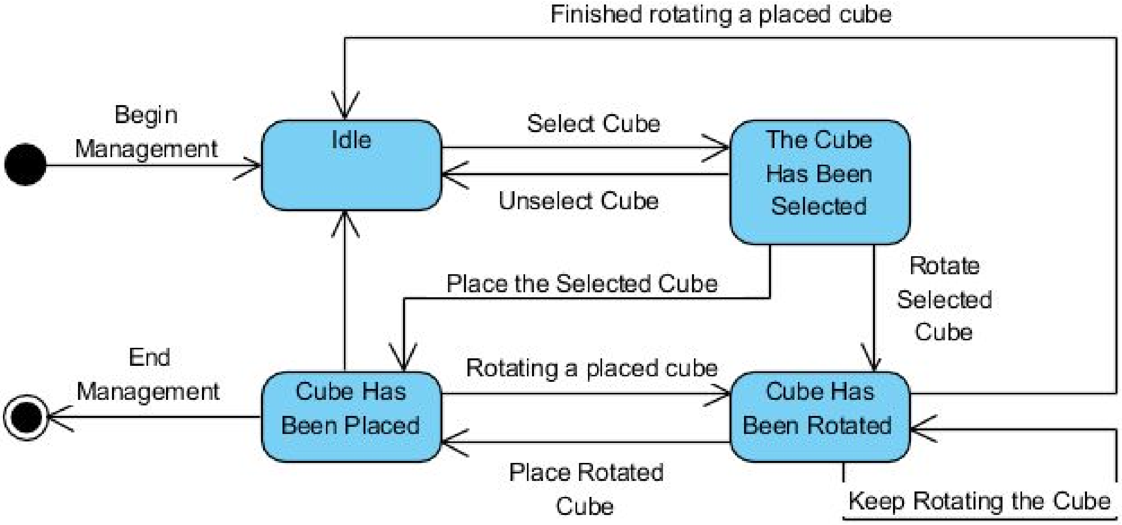


Figure 5.3.1 Cube Management State Diagram

Idle State: This state is when the cube is not selected at the beginning or the cube is unselected by the player or after a cube has been placed or placed.

The Cube Has Been Selected: A cube has been selected by player and it is ready to be rotated or to be placed.

The Cube Has Been Rotated: The selected cube has been rotated by the player to find the correct face.

The Cube Has Been Placed: At this state a cube has been placed on the board or a cube which is already on the board has been rotated. If the player is not done managing the cubes, player repeats the states (see Figure 5.3.1).

Classic Mode State Diagram

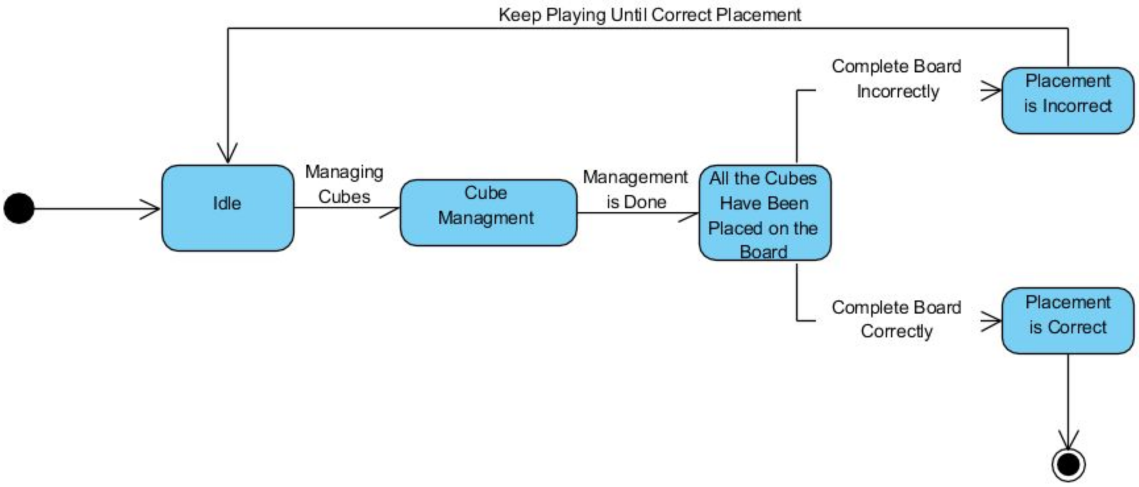


Figure 5.3.2 Classic Mode State Diagram

Idle State: This state is when no cubes are selected at the beginning of the game or when the placement is not correct.

Cube Management: In this state, the player manages the cubes to acquire the correct cube arrangement. Cube Management is a nested state (see Figure 5.3.1).

All the Cubes Have Been Placed on the Board: This is the state when a player is done with the cube management, and completed placing all the cubes on the board.

Placement is Incorrect: When the board is filled with cubes and the current arrangement is not the same with the game card, player repeats the states above and keeps playing until he/she gets the correct placement.

Placement is Correct: When the board is filled with cubes and the current arrangement matches with the game card game ends.

Switch Mode State Diagram

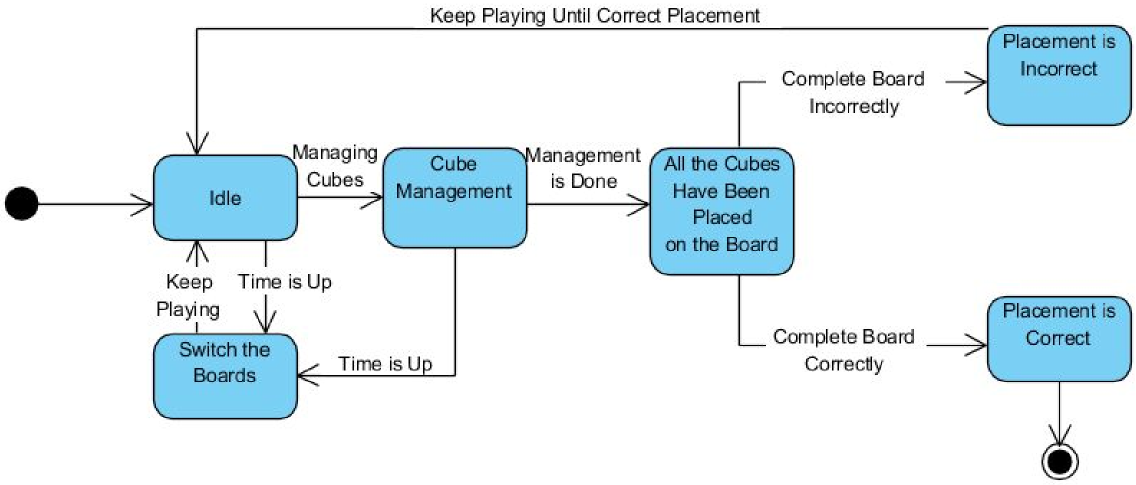


Figure 5.3.3 Switch Mode State Diagram

Idle State: This state is when no cubes are selected at the beginning of the game or when the boards has been switched after time is up or after a cube has been just placed.

Cube Management: In this state, the player manages the cubes to acquire the correct cube arrangement. Cube Management is a nested state (see Figure 5.3.1).

All the Cubes Have Been Placed on the Board: This is the state when a player is done with the cube management, and completed placing all the cubes on the board.

Switch the Boards: When time reserved for players to arrange or disarrange a board is up, the boards that player work on will switch with each other.

Placement is Incorrect: When the board is filled with cubes and the current arrangement is not the same with the game card, player repeats the states above and keeps playing until he/she gets the correct placement.

Placement is Correct: When the board is filled with cubes and the current arrangement matches with the game card game ends.

Survival Mode State Diagram

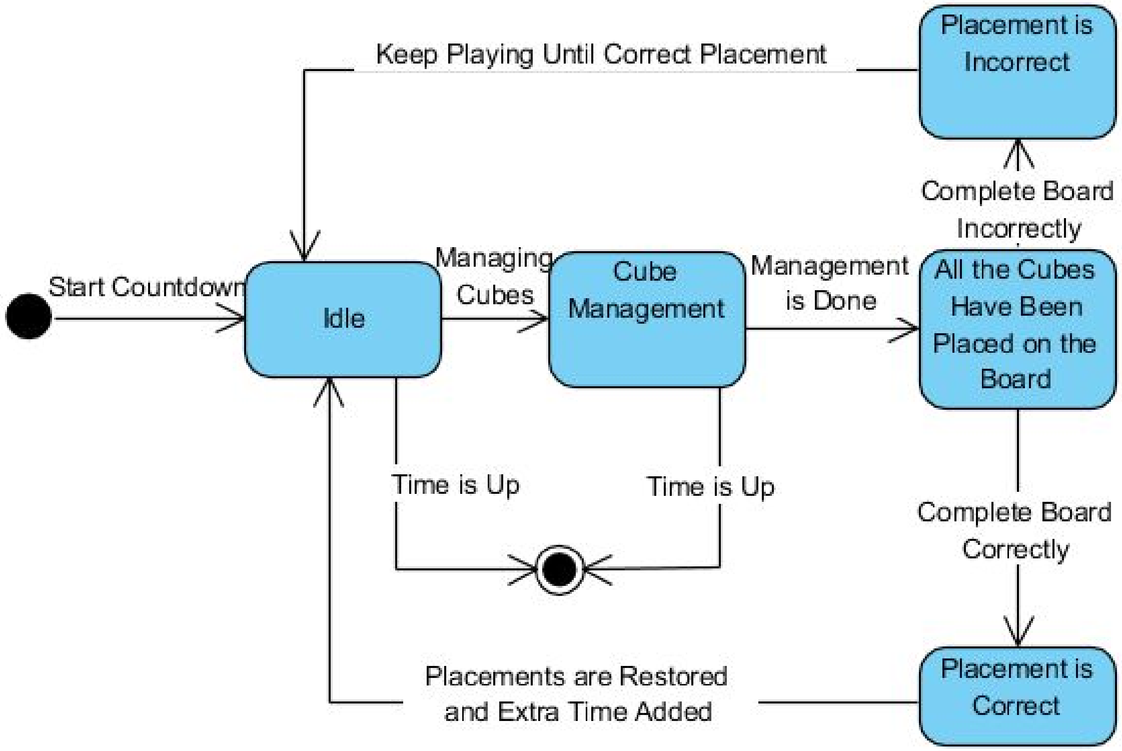


Figure 5.3.4 Survival Mode State Diagram

Idle State: This state is when no cubes are selected at the beginning of the game or after the current board has been completed.

Cube Management: In this state, the player manages the cubes to acquire the correct cube arrangement. Cube Management is a nested state (see Figure 5.3.1).

All the Cubes Have Been Placed on the Board: This is the state when a player is done with the cube management, and completed placing all the cubes on the board.

Placement is Incorrect: When the board is filled with cubes and the current arrangement is not the same with the game card, player repeats the states above and keeps playing until he/she gets the correct placement.

Placement is Correct: When the board is filled with cubes and the current arrangement matches with the game card a new game card appears, placement are restored and Bonus time is given to the player.

Daily Challenge Mode State Diagram

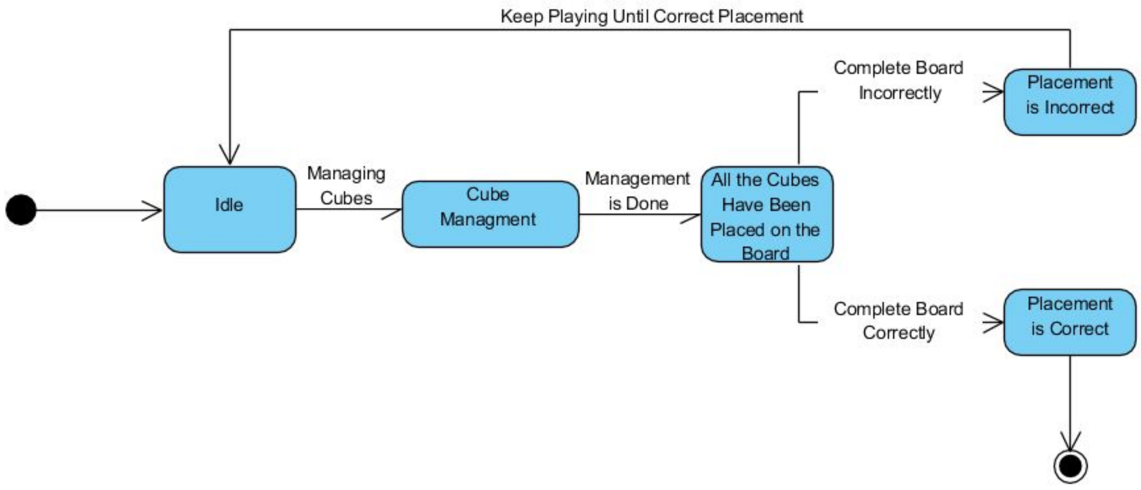


Figure 5.3.5 Daily Challenge Mode State Diagram

Idle State: This state is when no cubes are selected at the beginning of the game or when the placement is not correct.

Cube Management: In this state, the player manages the cubes to acquire the correct cube arrangement. Cube Management is a nested state (see Figure 5.3.1).

All the Cubes Have Been Placed on the Board: This is the state when a player is done with the cube management, and completed placing all the cubes on the board.

Placement is Incorrect: When the board is filled with cubes and the current arrangement is not the same with the game card, player repeats the states above and keeps playing until he/she gets the correct placement.

Placement is Correct: When the board is filled with cubes and the current arrangement matches with the game card game ends.

Sequence Diagrams

Survival Mode Sequence Diagram

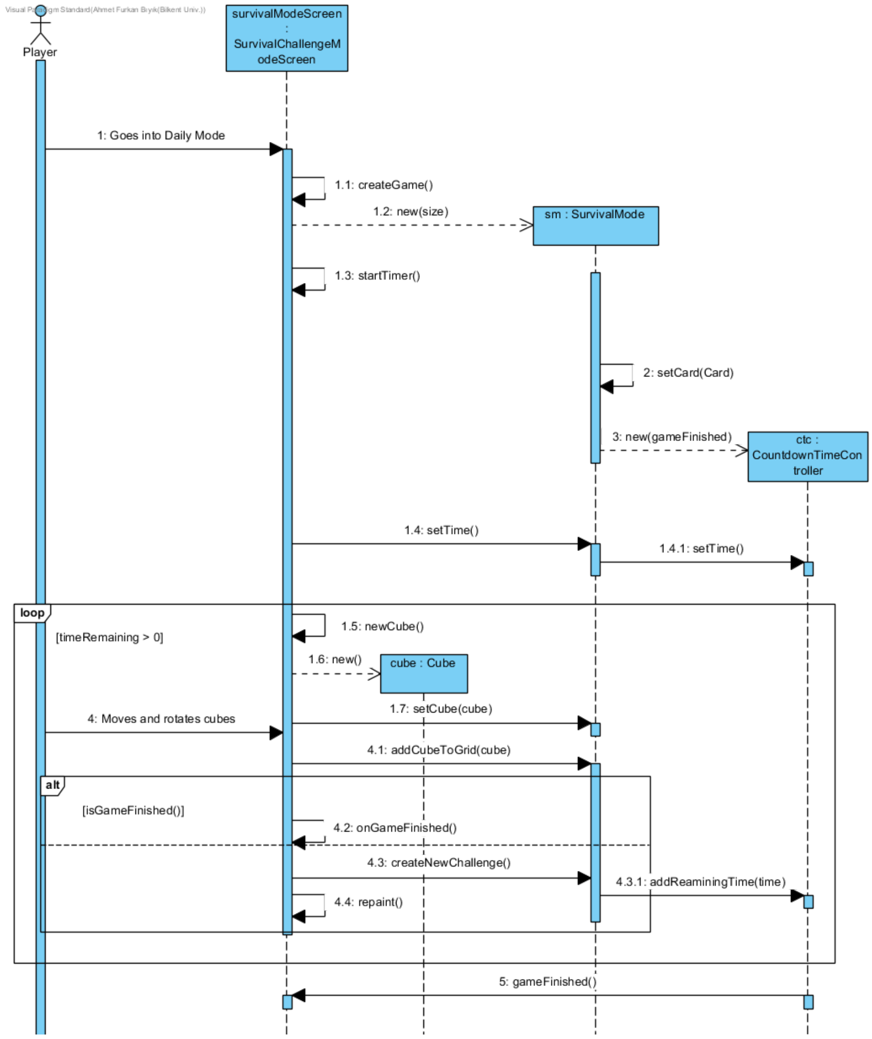


Figure 5.3.3 Survival Mode Sequence Diagram

In the survival mode, if player presses on an arrow key or places the cube, the cube will be moved to the desired place if he/she still has time. If current card is finished, additional time is added and new card, challenge is generated. If the game is over, the score will be displayed.

Daily Challenge Mode Sequence Diagram

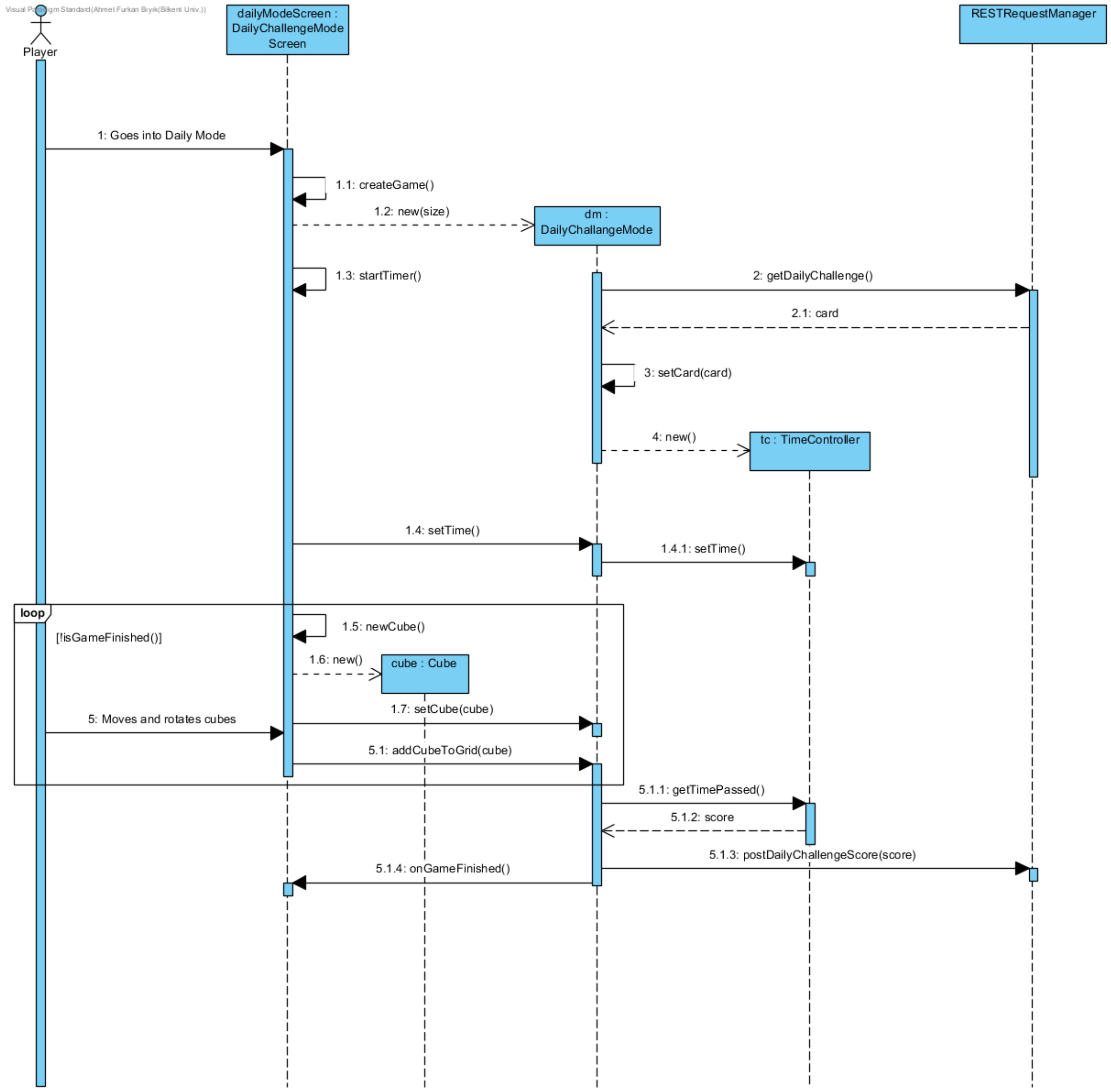


Figure 5.3.4 Daily Challenge Mode Sequence Diagram

In the daily challenge mode, if player presses on an arrow key or places the cube, the cube will be moved to the desired place. Card is gotten from server. If the overall placement of the board is correct, the next card will be assigned. When the game is over, the score – time elapsed – will be displayed and posted to server for leaderboard.

Classic Mode Sequence Diagram

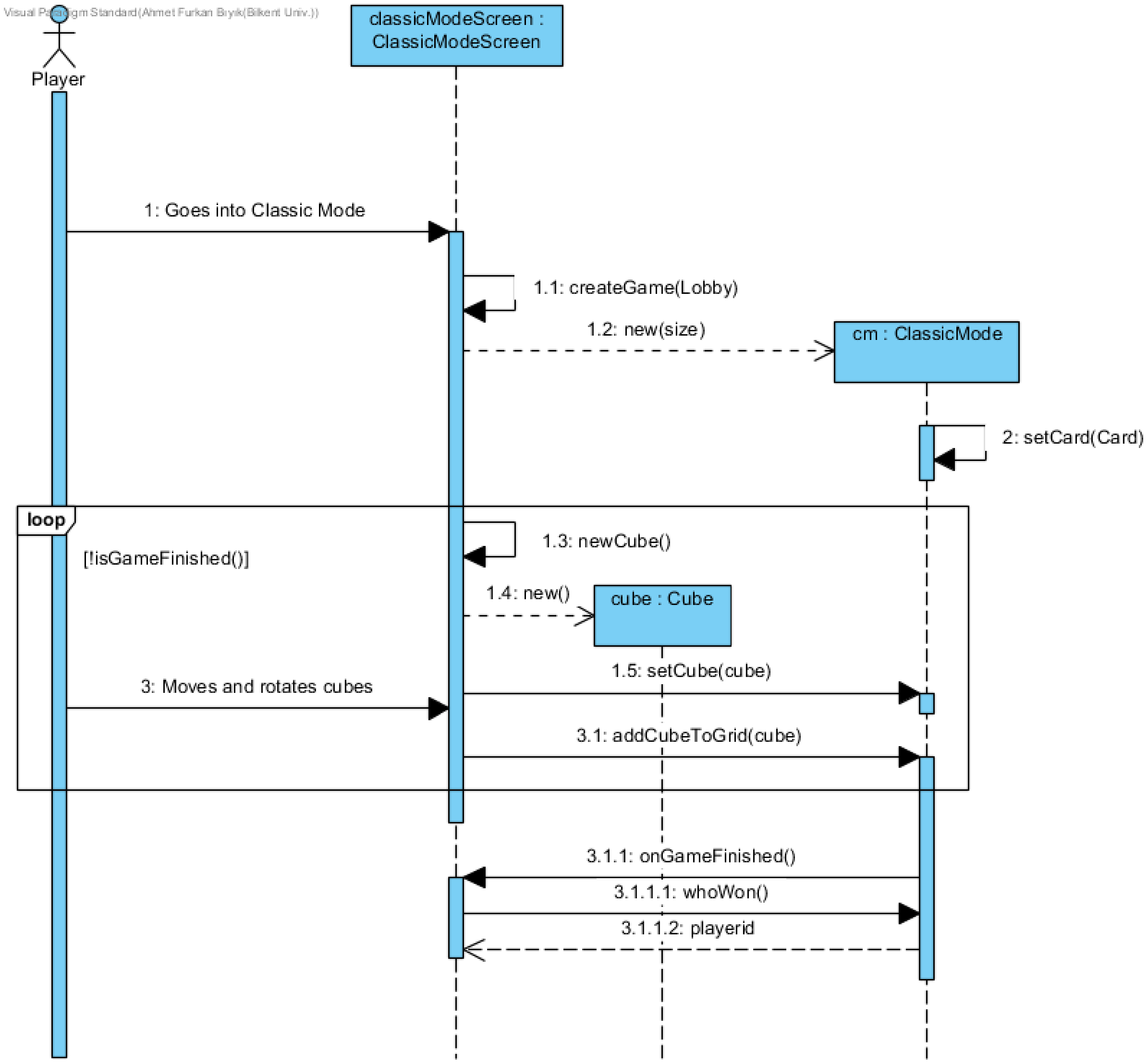


Figure 5.3.10 Classic Mode Sequence Diagram

In the classic mode, if player presses on an arrow key or places the cube, the cube will be moved to the desired place if the opponent has not completed his/her board. If the overall placement of the board is correct and the opponent is not finished yet, the player will be the winner of the round. When the game is over, winner of the game will be displayed.

Switch Mode Sequence Diagram

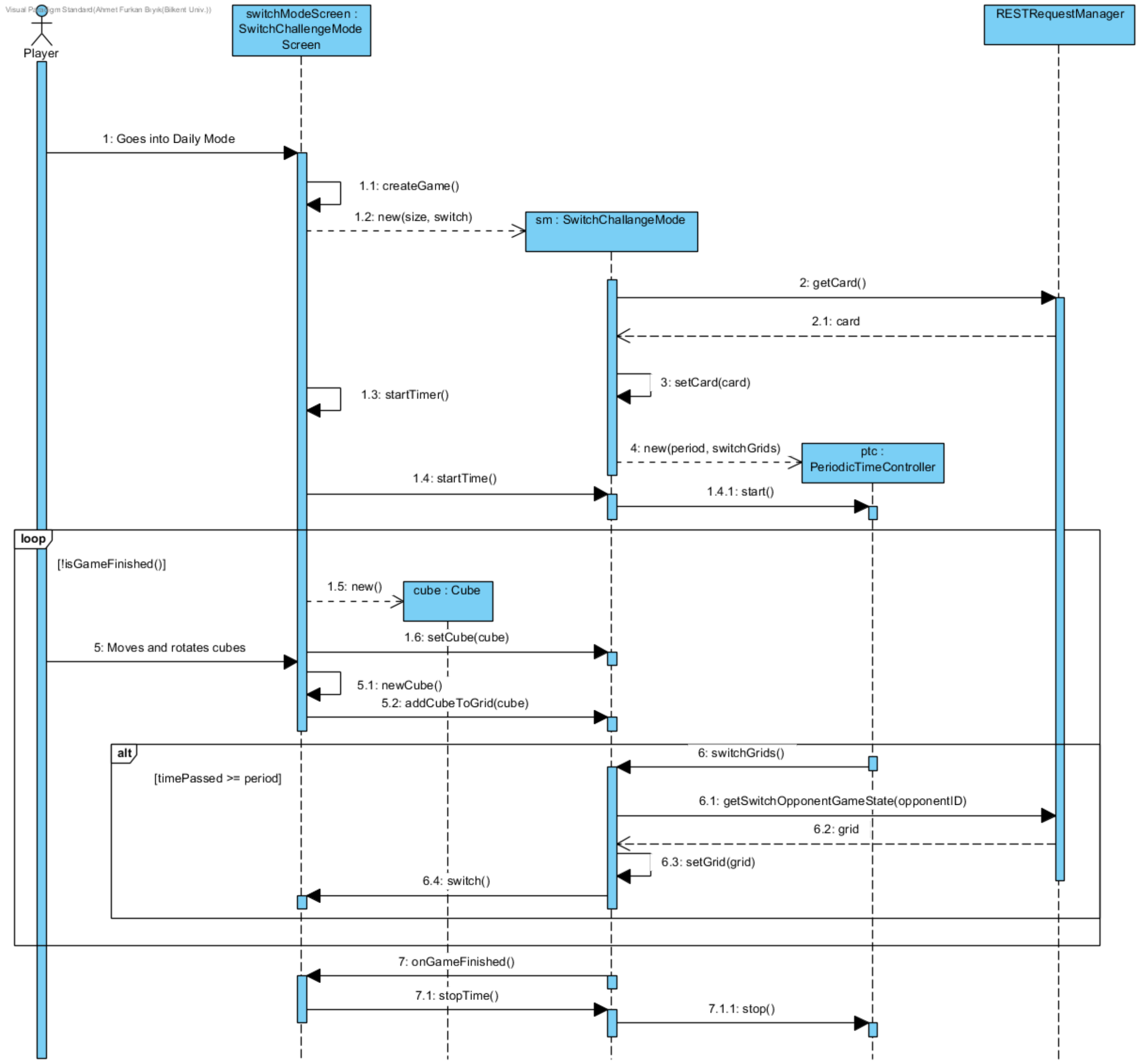


Figure 5.3.11 Switch Mode Sequence Diagram

In the switch mode, if player presses on an arrow key or places the cube, the cube will be moved to the desired place. In every predetermined time interval, they players will switch the boards – time will be controlled by a PeriodicTimeController –. If the overall placement of the board is correct, the game is over, and the winner will be displayed.

Object and Class Model

Logic Package Class Diagram:

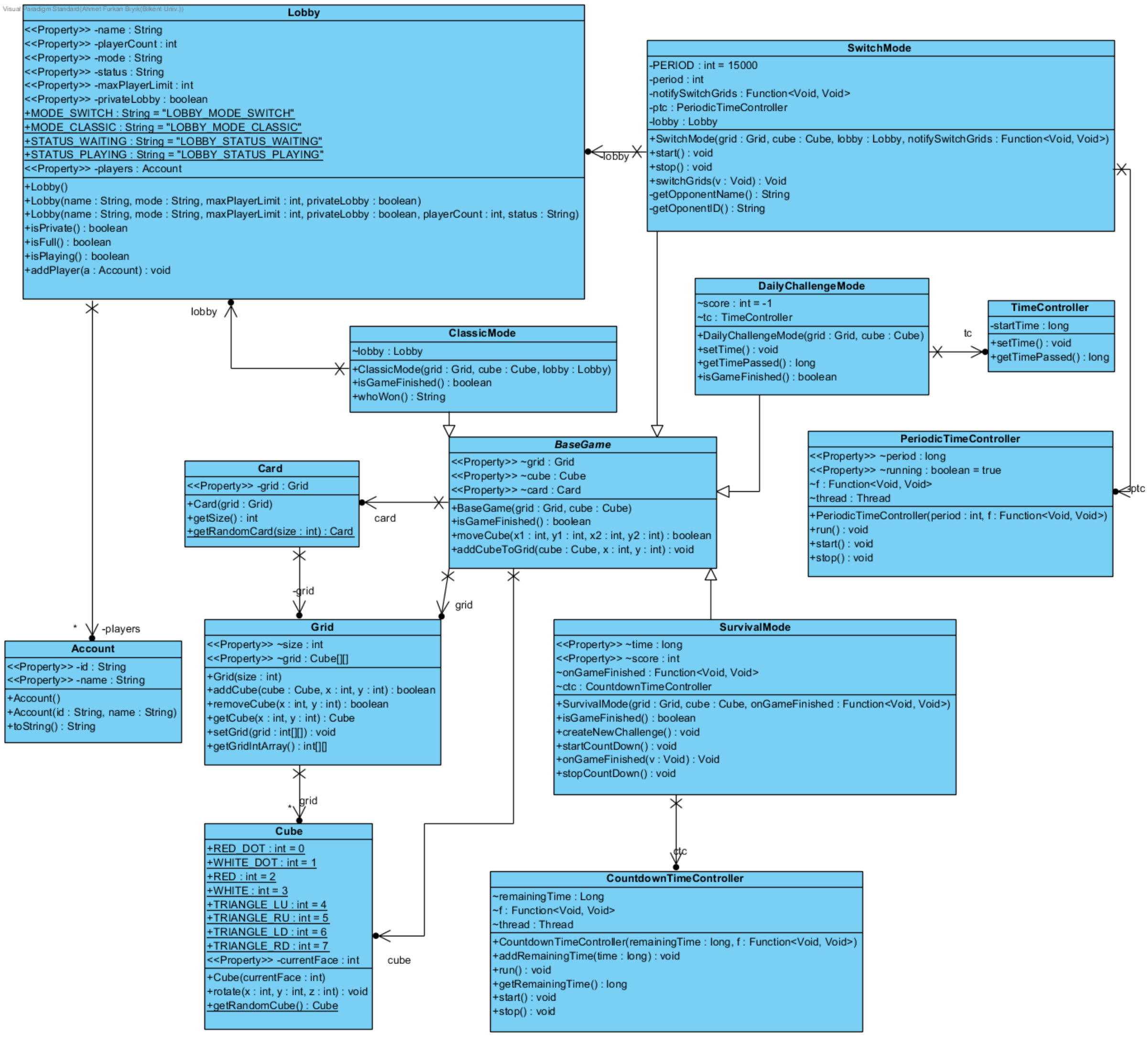
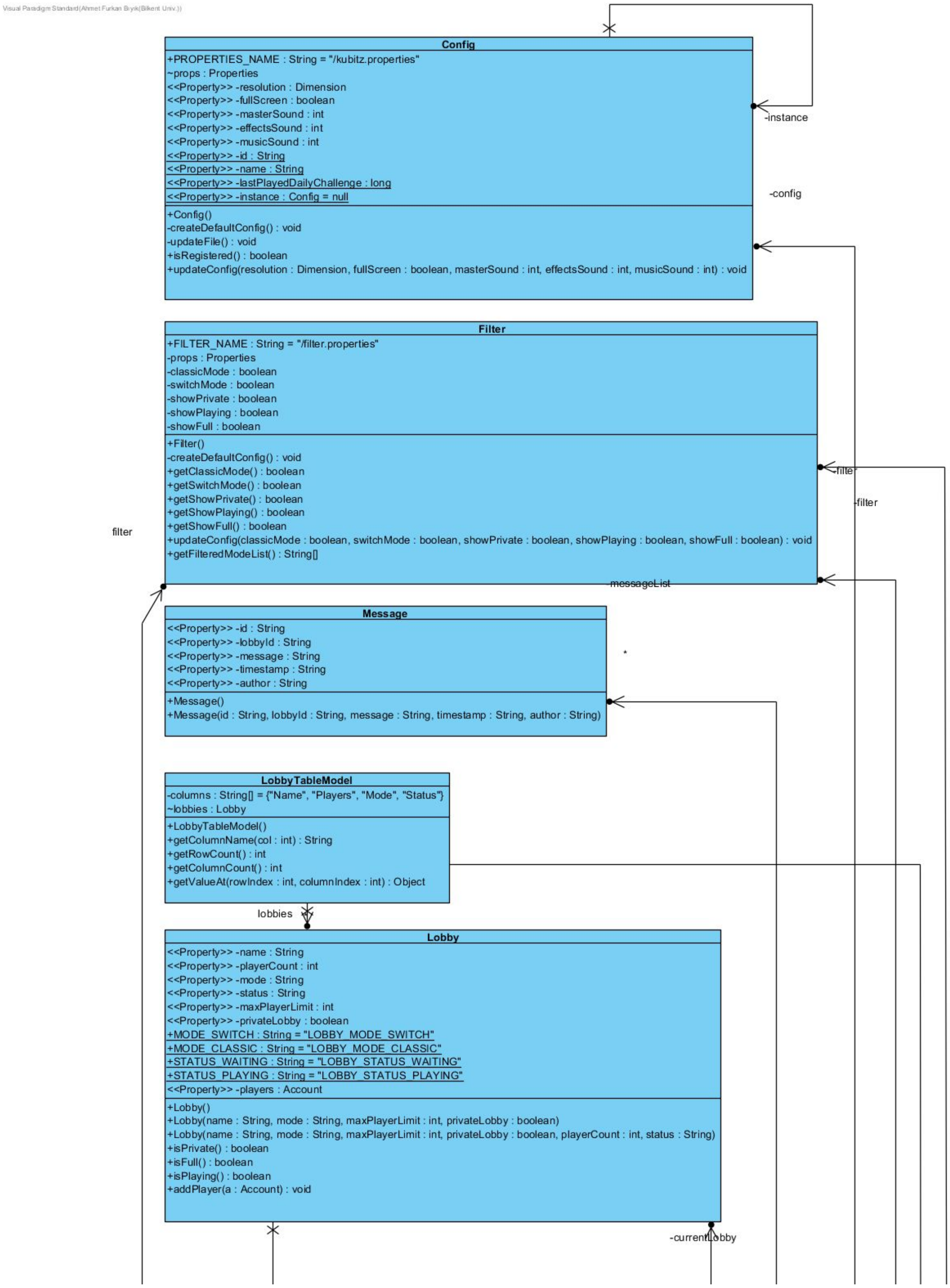
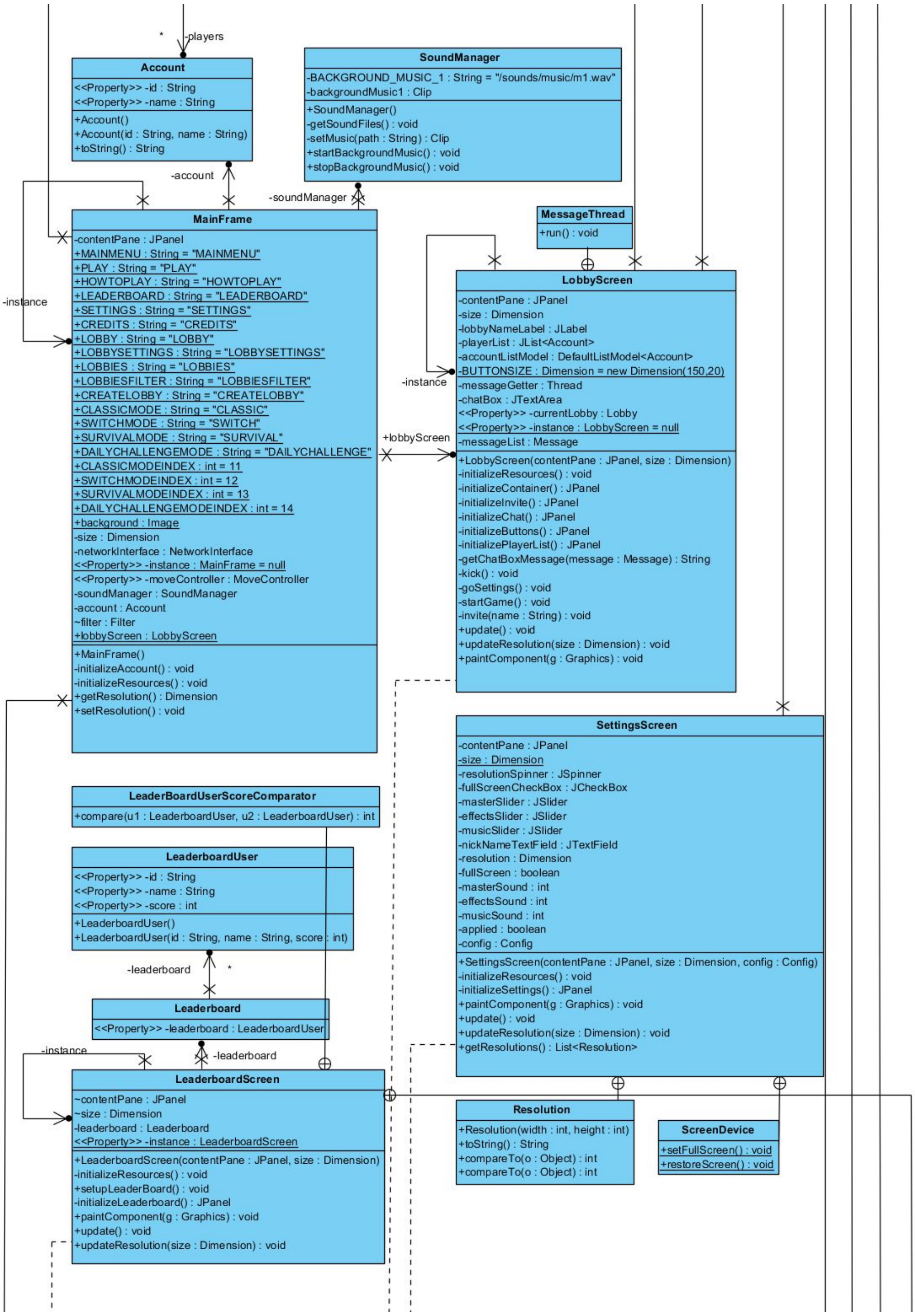
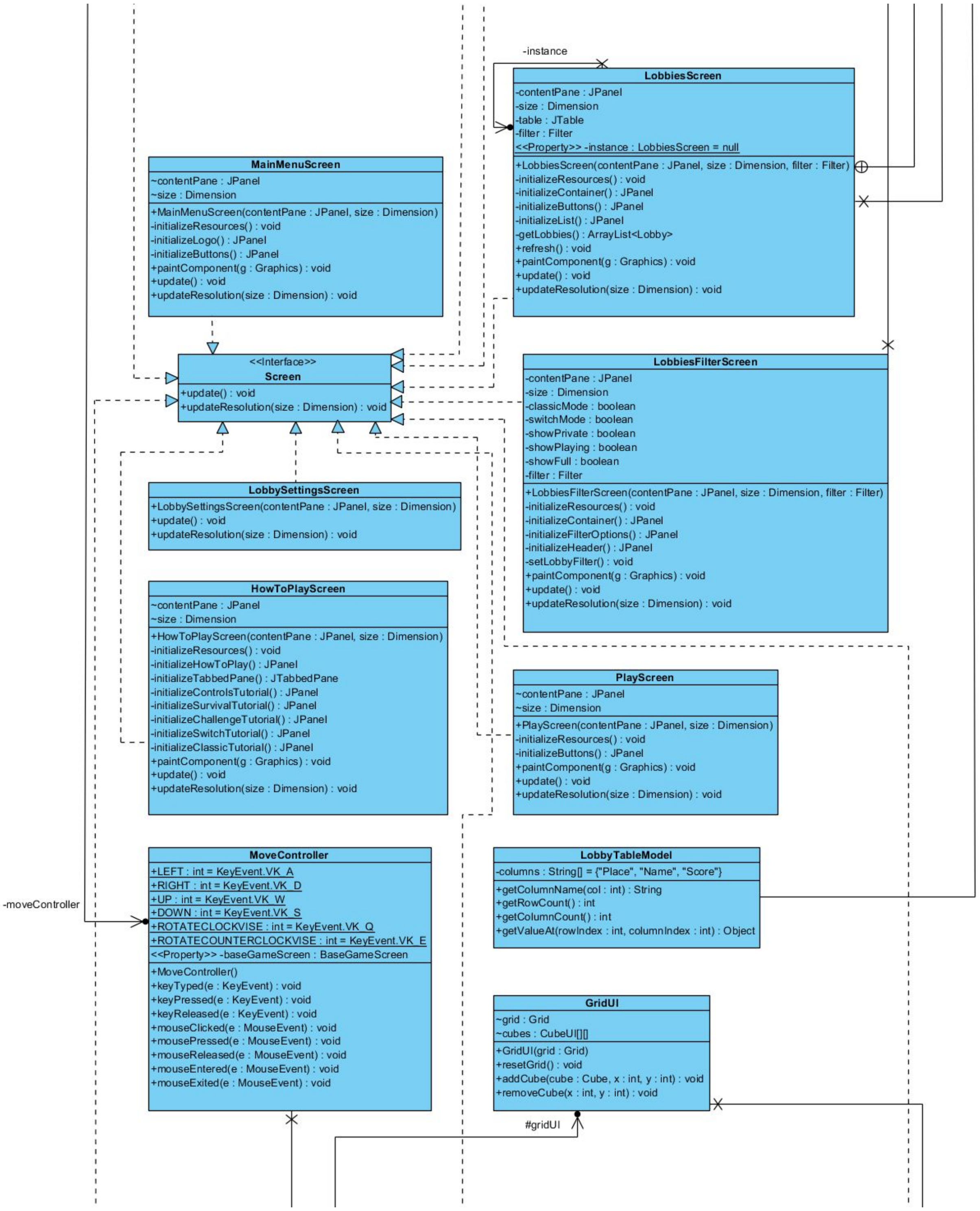

Visual Paradigm Standard(Ahmet Furkan Biyik(Bilkent Univ.))
Lobby
<<Property>> -name : String
<<Property>> -playerCount : int
<<Property>> -mode : String
<<Property>> -status : String
<<Property>> -maxPlayerLimit : int
<<Property>> -privateLobby : boolean
+MODE_SWITCH : String = "LOBBY_MODE_SWITCH"
+MODE_CLASSIC : String = "LOBBY_MODE_CLASSIC"
+STATUS_WAITING : String = "LOBBY_STATUS_WAITING"
+STATUS_PLAYING : String = "LOBBY_STATUS_PLAYING"
<<Property>> -players : Account
+Lobby()
+Lobby(name : String, mode : String, maxPlayerLimit : int, privateLobby : boolean)
+Lobby(name : String, mode : String, maxPlayerLimit : int, privateLobby : boolean, playerCount : int, status : String)
+isPrivate() : boolean
+isFull() : boolean
+isPlaying() : boolean
+addPlayer(a : Account) : void
-lobby
SwitchMode
-PERIOD : int = 15000
-period : int
-notifySwitchGrids : Function<Void, Void>
-ptc : PeriodicTimeController
-lobby : Lobby
+SwitchMode(grid : Grid, cube : Cube, lobby : Lobby, notifySwitchGrids : Function<Void, Void>)
+start() : void
+stop() : void
+switchGrids(v : Void) : Void
-getOpponentName() : String
-getOponentID() : String
lobby
ClassicMode
~lobby : Lobby
+ClassicMode(grid : Grid, cube : Cube, lobby : Lobby)
+isGameFinished() : boolean
+whoWon() : String
DailyChallengeMode
~score : int = -1
~tc : TimeController
+DailyChallengeMode(grid : Grid, cube : Cube)
+setTime() : void
+getTimePassed() : long
+isGameFinished() : boolean
tc
TimeController
-startTime : long
+setTime() : void
+getTimePassed() : long
PeriodicTimeController
<<Property>> ~period : long
<<Property>> ~running : boolean = true
~f : Function<Void, Void>
~thread : Thread
+PeriodicTimeController(period : int, f : Function<Void, Void>)
+run() : void
+start() : void
+stop() : void
-ptc
Card
<<Property>> -grid : Grid
+Card(grid : Grid)
+getSize() : int
+getRandomCard(size : int) : Card
card
BaseGame
<<Property>> ~grid : Grid
<<Property>> ~cube : Cube
<<Property>> ~card : Card
+BaseGame(grid : Grid, cube : Cube)
+isGameFinished() : boolean
+moveCube(x1 : int, y1 : int, x2 : int, y2 : int) : boolean
+addCubeToGrid(cube : Cube, x : int, y : int) : void
-grid
grid
*
-players
Account
<<Property>> -id : String
<<Property>> -name : String
+Account()
+Account(id : String, name : String)
+toString() : String
Grid
<<Property>> ~size : int
<<Property>> ~grid : Cube[][]
+Grid(size : int)
+addCube(cube : Cube, x : int, y : int) : boolean
+removeCube(x : int, y : int) : boolean
+getCube(x : int, y : int) : Cube
+setGrid(grid : int[][]) : void
+getGridIntArray() : int[][]
SurvivalMode
<<Property>> ~time : long
<<Property>> ~score : int
~onGameFinished : Function<Void, Void>
~ctc : CountdownTimeController
+SurvivalMode(grid : Grid, cube : Cube, onGameFinished : Function<Void, Void>)
+isGameFinished() : boolean
+createNewChallenge() : void
+startCountDown() : void
+onGameFinished(v : Void) : Void
+stopCountDown() : void
*
grid
Cube
+RED_DOT : int = 0
+WHITE_DOT : int = 1
+RED : int = 2
+WHITE : int = 3
+TRIANGLE_LU : int = 4
+TRIANGLE_RU : int = 5
+TRIANGLE_LD : int = 6
+TRIANGLE_RD : int = 7
<<Property>> -currentFace : int
+Cube(currentFace : int)
+rotate(x : int, y : int, z : int) : void
+getRandomCube() : Cube
cube
ctc
CountdownTimeController
~remainingTime : Long
~f : Function<Void, Void>
~thread : Thread
+CountdownTimeController(remainingTime : long, f : Function<Void, Void>)
+addRemainingTime(time : long) : void
+run() : void
+getRemainingTime() : long
+start() : void
+stop() : void

Figure 5.4.1 Logic Package Class Diagram

Gui Package Class Diagram:


Visual Paradigm Standard(Ahmet Furkan Biyik(Bilkent Univ.))
Config
+PROPERTIES_NAME : String = "/kubitz.properties"
~props : Properties
<<Property>> -resolution : Dimension
<<Property>> -fullScreen : boolean
<<Property>> -masterSound : int
<<Property>> -effectsSound : int
<<Property>> -musicSound : int
<<Property>> -id : String
<<Property>> -name : String
<<Property>> -lastPlayedDailyChallenge : long
<<Property>> -instance : Config = null
+Config()
-createDefaultConfig() : void
-updateFile() : void
+isRegistered() : boolean
+updateConfig(resolution : Dimension, fullScreen : boolean, masterSound : int, effectsSound : int, musicSound : int) : void
-instance
-config
Filter
+FILTER_NAME : String = "/filter.properties"
-props : Properties
-classicMode : boolean
-switchMode : boolean
-showPrivate : boolean
-showPlaying : boolean
-showFull : boolean
+Filter()
-createDefaultConfig() : void
+getClassicMode() : boolean
+getSwitchMode() : boolean
+getShowPrivate() : boolean
+getShowPlaying() : boolean
+getShowFull() : boolean
+updateConfig(classicMode : boolean, switchMode : boolean, showPrivate : boolean, showPlaying : boolean, showFull : boolean) : void
+getFilteredModeList() : String[]
-filte
-filter
filter
-messageList
Message
<<Property>> -id : String
<<Property>> -lobbyId : String
<<Property>> -message : String
<<Property>> -timestamp : String
<<Property>> -author : String
+Message()
+Message(id : String, lobbyId : String, message : String, timestamp : String, author : String)
*
LobbyTableModel
-columns : String[] = {"Name", "Players", "Mode", "Status"}
~lobbies : Lobby
+LobbyTableModel()
+getColumnName(col : int) : String
+getRowCount() : int
+getColumnCount() : int
+getValueAt(rowIndex : int, columnIndex : int) : Object
lobbies
Lobby
<<Property>> -name : String
<<Property>> -playerCount : int
<<Property>> -mode : String
<<Property>> -status : String
<<Property>> -maxPlayerLimit : int
<<Property>> -privateLobby : boolean
+MODE_SWITCH : String = "LOBBY_MODE_SWITCH"
+MODE_CLASSIC : String = "LOBBY_MODE_CLASSIC"
+STATUS_WAITING : String = "LOBBY_STATUS_WAITING"
+STATUS_PLAYING : String = "LOBBY_STATUS_PLAYING"
<<Property>> -players : Account
+Lobby()
+Lobby(name : String, mode : String, maxPlayerLimit : int, privateLobby : boolean)
+Lobby(name : String, mode : String, maxPlayerLimit : int, privateLobby : boolean, playerCount : int, status : String)
+isPrivate() : boolean
+isFull() : boolean
+isPlaying() : boolean
+addPlayer(a : Account) : void
-currentLobby


*
-players
Account
<<Property>> -id : String
<<Property>> -name : String
+Account()
+Account(id : String, name : String)
+toString() : String
-account
SoundManager
-BACKGROUND_MUSIC_1 : String = "/sounds/music/m1.wav"
-backgroundMusic1 : Clip
+SoundManager()
-getSoundFiles() : void
-setMusic(path : String) : Clip
+startBackgroundMusic() : void
+stopBackgroundMusic() : void
-soundManager
-instance
MainFrame
-contentPane : JPanel
+MAINMENU : String = "MAINMENU"
+PLAY : String = "PLAY"
+HOWTOPLAY : String = "HOWTOPLAY"
+LEADERBOARD : String = "LEADERBOARD"
+SETTINGS : String = "SETTINGS"
+CREDITS : String = "CREDITS"
+LOBBY : String = "LOBBY"
+LOBBYSETTINGS : String = "LOBBYSETTINGS"
+LOBBIES : String = "LOBBIES"
+LOBBIESFILTER : String = "LOBBIESFILTER"
+CREATELOBBY : String = "CREATELOBBY"
+CLASSICMODE : String = "CLASSIC"
+SWITCHMODE : String = "SWITCH"
+SURVIVALMODE : String = "SURVIVAL"
+DAILYCHALLENGEMODE : String = "DAILYCHALLENGE"
+CLASSICMODEINDEX : int = 11
+SWITCHMODEINDEX : int = 12
+SURVIVALMODEINDEX : int = 13
+DAILYCHALLENGEMODEINDEX : int = 14
+background : Image
-size : Dimension
-networkInterface : NetworkInterface
<<Property>> -instance : MainFrame = null
<<Property>> -moveController : MoveController
-soundManager : SoundManager
-account : Account
~filter : Filter
+lobbyScreen : LobbyScreen
+MainFrame()
-initializeAccount() : void
-initializeResources() : void
+getResolution() : Dimension
+setResolution() : void
MessageThread
+run() : void
-instance
+lobbyScreen
LobbyScreen
-contentPane : JPanel
-size : Dimension
-lobbyNameLabel : JLabel
-playerList : JList<Account>
-accountListModel : DefaultListModel<Account>
-BUTTONSIZE : Dimension = new Dimension(150,20)
-messageGetter : Thread
-chatBox : JTextArea
<<Property>> -currentLobby : Lobby
<<Property>> -instance : LobbyScreen = null
-messageList : Message
+LobbyScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeContainer() : JPanel
-initializeInvite() : JPanel
-initializeChat() : JPanel
-initializeButtons() : JPanel
-initializePlayerList() : JPanel
-getChatBoxMessage(message : Message) : String
-kick() : void
-goSettings() : void
-startGame() : void
-invite(name : String) : void
+update() : void
+updateResolution(size : Dimension) : void
+paintComponent(g : Graphics) : void
SettingsScreen
-contentPane : JPanel
-size : Dimension
-resolutionSpinner : JSpinner
-fullScreenCheckBox : JCheckBox
-masterSlider : JSlider
-effectsSlider : JSlider
-musicSlider : JSlider
-nickNameTextField : JTextField
-resolution : Dimension
-fullScreen : boolean
-masterSound : int
-effectsSound : int
-musicSound : int
-applied : boolean
-config : Config
+SettingsScreen(contentPane : JPanel, size : Dimension, config : Config)
-initializeResources() : void
-initializeSettings() : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
+getResolutions() : List<Resolution>
LeaderBoardUserScoreComparator
+compare(u1 : LeaderboardUser, u2 : LeaderboardUser) : int
LeaderboardUser
<<Property>> -id : String
<<Property>> -name : String
<<Property>> -score : int
+LeaderboardUser()
+LeaderboardUser(id : String, name : String, score : int)
-leaderboard
*
Leaderboard
<<Property>> -leaderboard : LeaderboardUser
-instance
-leaderboard
LeaderboardScreen
~contentPane : JPanel
~size : Dimension
-leaderboard : Leaderboard
<<Property>> -instance : LeaderboardScreen
+LeaderboardScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
+setupLeaderBoard() : void
-initializeLeaderboard() : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
Resolution
+Resolution(width : int, height : int)
+toString() : String
+compareTo(o : Object) : int
+compareTo(o : Object) : int
ScreenDevice
+setFullScreen() : void
+restoreScreen() : void


-instance
LobbiesScreen
-contentPane : JPanel
-size : Dimension
-table : JTable
-filter : Filter
<<Property>> -instance : LobbiesScreen = null
+LobbiesScreen(contentPane : JPanel, size : Dimension, filter : Filter)
-initializeResources() : void
-initializeContainer() : JPanel
-initializeButtons() : JPanel
-initializeList() : JPanel
-getLobbies() : ArrayList<Lobby>
+refresh() : void
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
MainMenuScreen
~contentPane : JPanel
~size : Dimension
+MainMenuScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeLogo() : JPanel
-initializeButtons() : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
<<Interface>>
Screen
+update() : void
+updateResolution(size : Dimension) : void
LobbiesFilterScreen
-contentPane : JPanel
-size : Dimension
-classicMode : boolean
-switchMode : boolean
-showPrivate : boolean
-showPlaying : boolean
-showFull : boolean
-filter : Filter
+LobbiesFilterScreen(contentPane : JPanel, size : Dimension, filter : Filter)
-initializeResources() : void
-initializeContainer() : JPanel
-initializeFilterOptions() : JPanel
-initializeHeader() : JPanel
-setLobbyFilter() : void
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
LobbySettingsScreen
+LobbySettingsScreen(contentPane : JPanel, size : Dimension)
+update() : void
+updateResolution(size : Dimension) : void
HowToPlayScreen
~contentPane : JPanel
~size : Dimension
+HowToPlayScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeHowToPlay() : JPanel
-initializeTabbedPane() : JTabbedPane
-initializeControlsTutorial() : JPanel
-initializeSurvivalTutorial() : JPanel
-initializeChallengeTutorial() : JPanel
-initializeSwitchTutorial() : JPanel
-initializeClassicTutorial() : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
PlayScreen
~contentPane : JPanel
~size : Dimension
+PlayScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeButtons() : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
-moveController
MoveController
+LEFT : int = KeyEvent.VK_A
+RIGHT : int = KeyEvent.VK_D
+UP : int = KeyEvent.VK_W
+DOWN : int = KeyEvent.VK_S
+ROTATECLOCKVISE : int = KeyEvent.VK_Q
+ROTATECOUNTERCLOCKVISE : int = KeyEvent.VK_E
<<Property>> -baseGameScreen : BaseGameScreen
+MoveController()
+keyTyped(e : KeyEvent) : void
+keyPressed(e : KeyEvent) : void
+keyReleased(e : KeyEvent) : void
+mouseClicked(e : MouseEvent) : void
+mousePressed(e : MouseEvent) : void
+mouseReleased(e : MouseEvent) : void
+mouseEntered(e : MouseEvent) : void
+mouseExited(e : MouseEvent) : void
LobbyTableModel
-columns : String[] = {"Place", "Name", "Score"}
+getColumnName(col : int) : String
+getRowCount() : int
+getColumnCount() : int
+getValueAt(rowIndex : int, columnIndex : int) : Object
GridUI
~grid : Grid
~cubes : CubeUI[][]
+GridUI(grid : Grid)
+resetGrid() : void
+addCube(cube : Cube, x : int, y : int) : void
+removeCube(x : int, y : int) : void
#gridUI

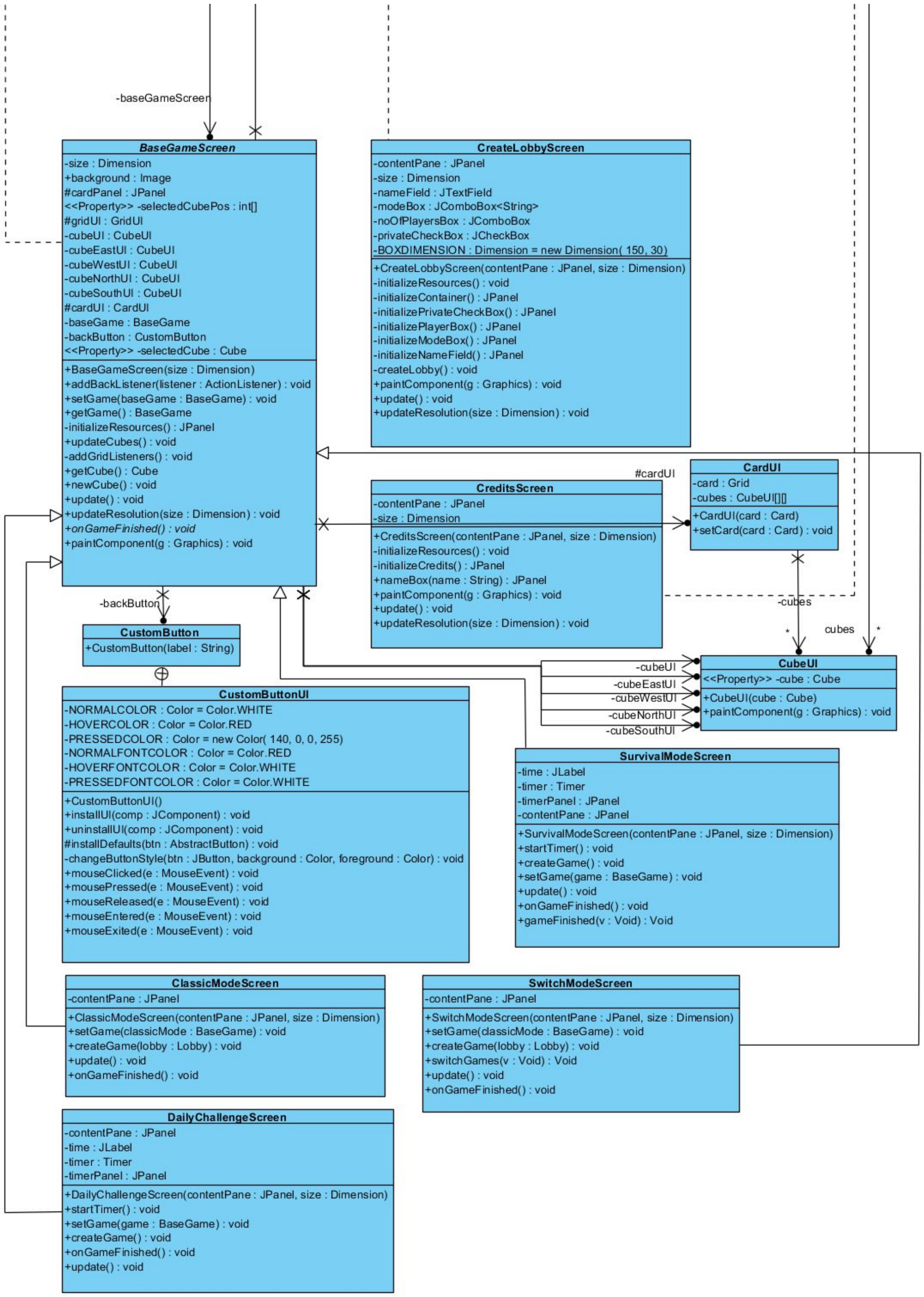
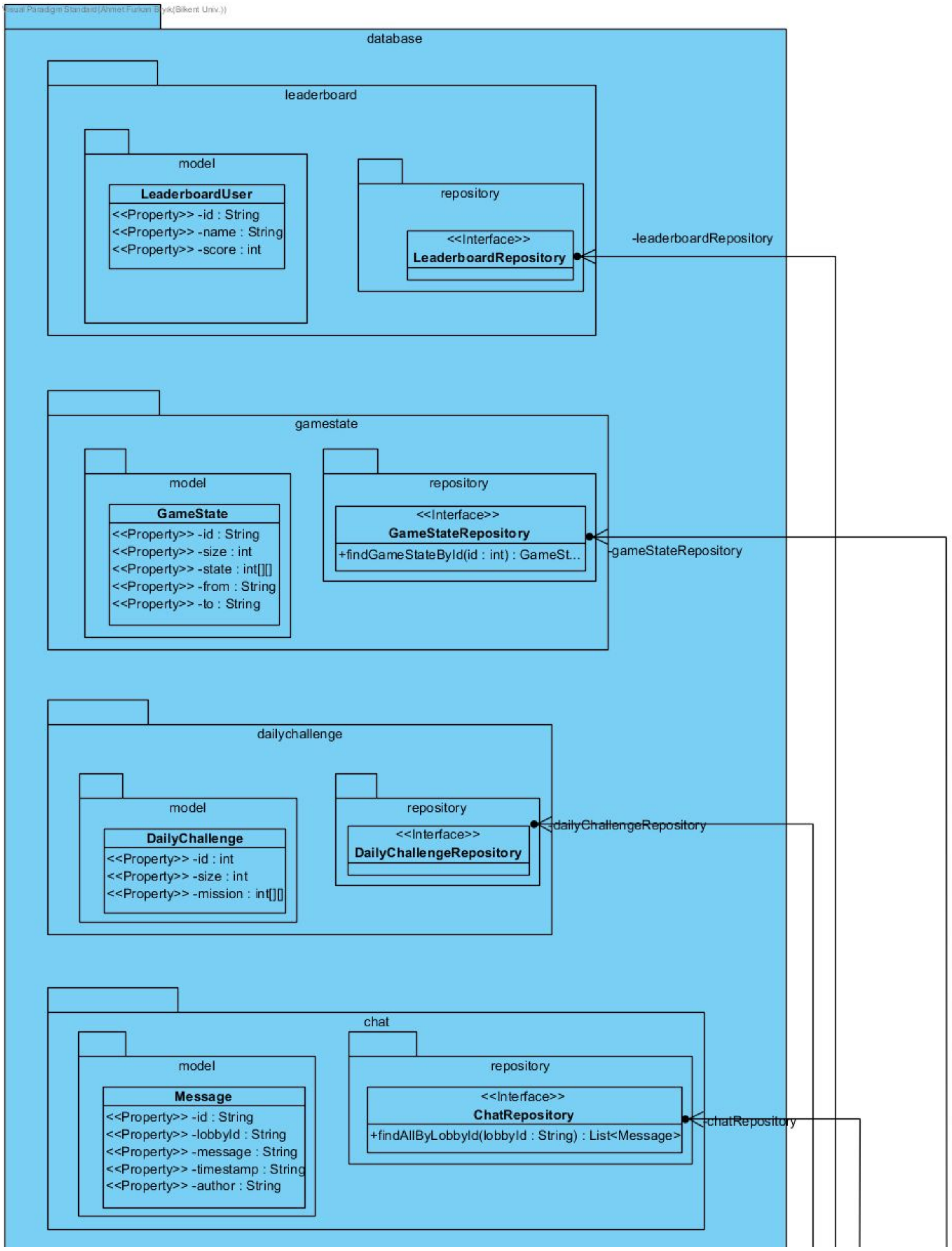
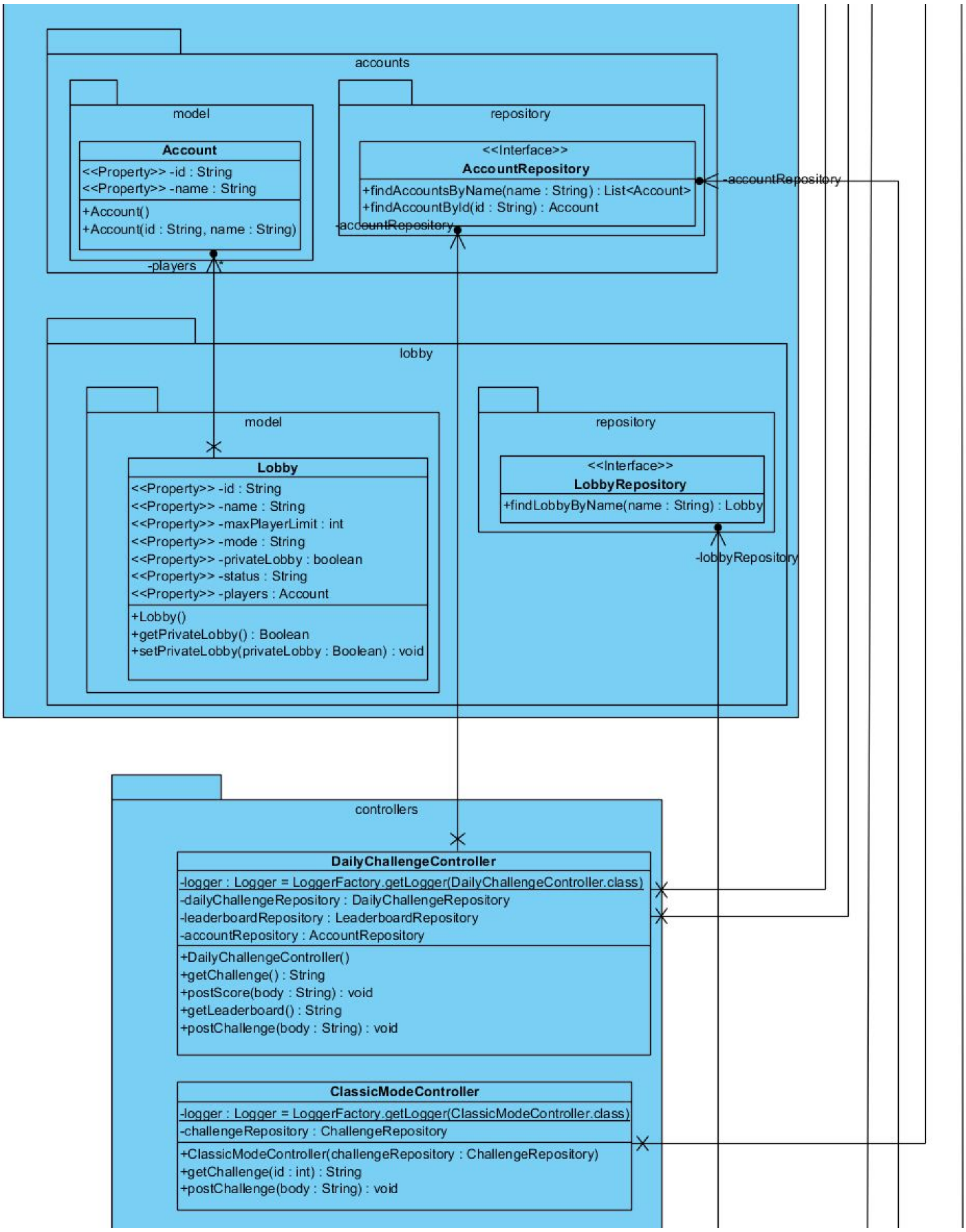

-baseGameScreen
BaseGameScreen
-size : Dimension
+background : Image
#cardPanel : JPanel
<<Property>> -selectedCubePos : int[]
#gridUI : GridUI
-cubeUI : CubeUI
-cubeEastUI : CubeUI
-cubeWestUI : CubeUI
-cubeNorthUI : CubeUI
-cubeSouthUI : CubeUI
#cardUI : CardUI
-baseGame : BaseGame
-backButton : CustomButton
<<Property>> -selectedCube : Cube
+BaseGameScreen(size : Dimension)
+addBackListener(listener : ActionListener) : void
+setGame(baseGame : BaseGame) : void
+getGame() : BaseGame
-initializeResources() : JPanel
+updateCubes() : void
-addGridListeners() : void
+getCube() : Cube
+newCube() : void
+update() : void
+updateResolution(size : Dimension) : void
+onGameFinished() : void
+paintComponent(g : Graphics) : void
CreateLobbyScreen
-contentPane : JPanel
-size : Dimension
-nameField : JTextField
-modeBox : JComboBox<String>
-noOfPlayersBox : JComboBox
-privateCheckBox : JCheckBox
-BOXDIMENSION : Dimension = new Dimension(150, 30)
+CreateLobbyScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeContainer() : JPanel
-initializePrivateCheckBox() : JPanel
-initializePlayerBox() : JPanel
-initializeModeBox() : JPanel
-initializeNameField() : JPanel
-createLobby() : void
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
#cardUI
CardUI
-card : Grid
-cubes : CubeUI[][]
+CardUI(card : Card)
+setCard(card : Card) : void
CreditsScreen
-contentPane : JPanel
-size : Dimension
+CreditsScreen(contentPane : JPanel, size : Dimension)
-initializeResources() : void
-initializeCredits() : JPanel
+nameBox(name : String) : JPanel
+paintComponent(g : Graphics) : void
+update() : void
+updateResolution(size : Dimension) : void
-cubes
*
cubes
*
-backButton
CustomButton
+CustomButton(label : String)
CustomButtonUI
-NORMALCOLOR : Color = Color.WHITE
-HOVERCOLOR : Color = Color.RED
-PRESSEDCOLOR : Color = new Color(140, 0, 0, 255)
-NORMALFONTCOLOR : Color = Color.RED
-HOVERFONTCOLOR : Color = Color.WHITE
-PRESSEDFONTCOLOR : Color = Color.WHITE
+CustomButtonUI()
+installUI(comp : JComponent) : void
+uninstallUI(comp : JComponent) : void
#installDefaults(btn : AbstractButton) : void
-changeButtonStyle(btn : JButton, background : Color, foreground : Color) : void
+mouseClicked(e : MouseEvent) : void
+mousePressed(e : MouseEvent) : void
+mouseReleased(e : MouseEvent) : void
+mouseEntered(e : MouseEvent) : void
+mouseExited(e : MouseEvent) : void
-cubeUI
-cubeEastUI
-cubeWestUI
-cubeNorthUI
-cubeSouthUI
CubeUI
<<Property>> -cube : Cube
+CubeUI(cube : Cube)
+paintComponent(g : Graphics) : void
SurvivalModeScreen
-time : JLabel
-timer : Timer
-timerPanel : JPanel
-contentPane : JPanel
+SurvivalModeScreen(contentPane : JPanel, size : Dimension)
+startTimer() : void
+createGame() : void
+setGame(game : BaseGame) : void
+update() : void
+onGameFinished() : void
+gameFinished(v : Void) : Void
ClassicModeScreen
-contentPane : JPanel
+ClassicModeScreen(contentPane : JPanel, size : Dimension)
+setGame(classicMode : BaseGame) : void
+createGame(lobby : Lobby) : void
+update() : void
+onGameFinished() : void
SwitchModeScreen
-contentPane : JPanel
+SwitchModeScreen(contentPane : JPanel, size : Dimension)
+setGame(classicMode : BaseGame) : void
+createGame(lobby : Lobby) : void
+switchGames(v : Void) : Void
+update() : void
+onGameFinished() : void
DailyChallengeScreen
-contentPane : JPanel
-time : JLabel
-timer : Timer
-timerPanel : JPanel
+DailyChallengeScreen(contentPane : JPanel, size : Dimension)
+startTimer() : void
+setGame(game : BaseGame) : void
+createGame() : void
+onGameFinished() : void
+update() : void

Figure 5.4.1 Gui Package Class Diagram

Server Class Diagram:


database
leaderboard
model
LeaderboardUser
<<Property>> -id : String
<<Property>> -name : String
<<Property>> -score : int
repository
<<Interface>>
LeaderboardRepository
-leaderboardRepository
gamestate
model
GameState
<<Property>> -id : String
<<Property>> -size : int
<<Property>> -state : int[][]
<<Property>> -from : String
<<Property>> -to : String
repository
<<Interface>>
GameStateRepository
+findGameStateById(id : int) : GameSt...
-gameStateRepository
dailychallenge
model
DailyChallenge
<<Property>> -id : int
<<Property>> -size : int
<<Property>> -mission : int[][]
repository
<<Interface>>
DailyChallengeRepository
-dailyChallengeRepository
chat
model
Message
<<Property>> -id : String
<<Property>> -lobbyId : String
<<Property>> -message : String
<<Property>> -timestamp : String
<<Property>> -author : String
repository
<<Interface>>
ChatRepository
+findAllByLobbyId(lobbyId : String) : List<Message>
-chatRepository


accounts
model
Account
<<Property>> -id : String
<<Property>> -name : String
+Account()
+Account(id : String, name : String)
repository
<<Interface>>
AccountRepository
+findAccountsByName(name : String) : List<Account>
+findAccountById(id : String) : Account
-accountRepository
-accountRepository
-players
*
lobby
model
Lobby
<<Property>> -id : String
<<Property>> -name : String
<<Property>> -maxPlayerLimit : int
<<Property>> -mode : String
<<Property>> -privateLobby : boolean
<<Property>> -status : String
<<Property>> -players : Account
+Lobby()
+getPrivateLobby() : Boolean
+setPrivateLobby(privateLobby : Boolean) : void
repository
<<Interface>>
LobbyRepository
+findLobbyByName(name : String) : Lobby
-lobbyRepository
controllers
DailyChallengeController
-logger : Logger = LoggerFactory.getLogger(DailyChallengeController.class)
-dailyChallengeRepository : DailyChallengeRepository
-leaderboardRepository : LeaderboardRepository
-accountRepository : AccountRepository
+DailyChallengeController()
+getChallenge() : String
+postScore(body : String) : void
+getLeaderboard() : String
+postChallenge(body : String) : void
ClassicModeController
-logger : Logger = LoggerFactory.getLogger(ClassicModeController.class)
-challengeRepository : ChallengeRepository
+ClassicModeController(challengeRepository : ChallengeRepository)
+getChallenge(id : int) : String
+postChallenge(body : String) : void

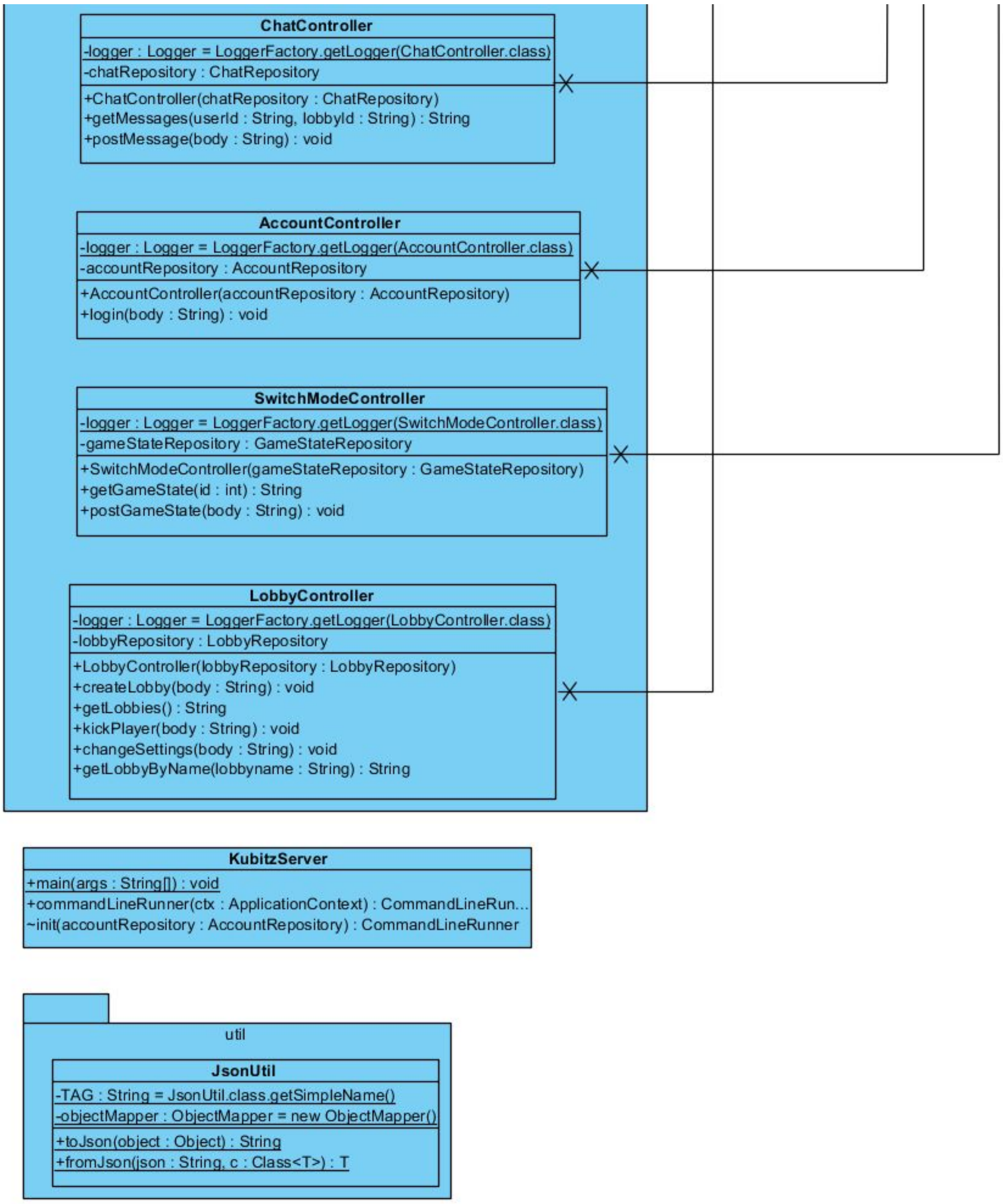

ChatController
-logger : Logger = LoggerFactory.getLogger(ChatController.class)
-chatRepository : ChatRepository
+ChatController(chatRepository : ChatRepository)
+getMessages(userId : String, lobbyId : String) : String
+postMessage(body : String) : void
AccountController
-logger : Logger = LoggerFactory.getLogger(AccountController.class)
-accountRepository : AccountRepository
+AccountController(accountRepository : AccountRepository)
+login(body : String) : void
SwitchModeController
-logger : Logger = LoggerFactory.getLogger(SwitchModeController.class)
-gameStateRepository : GameStateRepository
+SwitchModeController(gameStateRepository : GameStateRepository)
+getGameState(id : int) : String
+postGameState(body : String) : void
LobbyController
-logger : Logger = LoggerFactory.getLogger(LobbyController.class)
-lobbyRepository : LobbyRepository
+LobbyController(lobbyRepository : LobbyRepository)
+createLobby(body : String) : void
+getLobbies() : String
+kickPlayer(body : String) : void
+changeSettings(body : String) : void
+getLobbyByName(lobbyname : String) : String
KubitzServer
+main(args : String[]) : void
+commandLineRunner(ctx : ApplicationContext) : CommandLineRun...
~init(accountRepository : AccountRepository) : CommandLineRunner
util
JsonUtil
-TAG : String = JsonUtil.class.getSimpleName()
-objectMapper : ObjectMapper = new ObjectMapper()
+toJson(object : Object) : String
+fromJson(json : String, c : Class<T>) : T

Figure 5.4.1 Server Class Diagram

Improvement Summary

There were some functional requirements which we were not determined about the implementation and design choice of it. For instance, we were between providing a keyboard interface or mouse interface or even both of them to the player to control the game board and cubes. We first decided to provide both of them, however, we changed our mind and decided to make some experiments on players and analyze finish times as well as the player's feedbacks and finally, make the choice depending on the results of the experiments. We are planning to apply this approach when we are indeterminate about a requirement.

We also decided to be non-restrictive about the key bindings and added a settings option for arranging key bindings according to the personal preference of the player.

Other than the design changes above, we are consistent about our choices and produced our roadmap as follows.

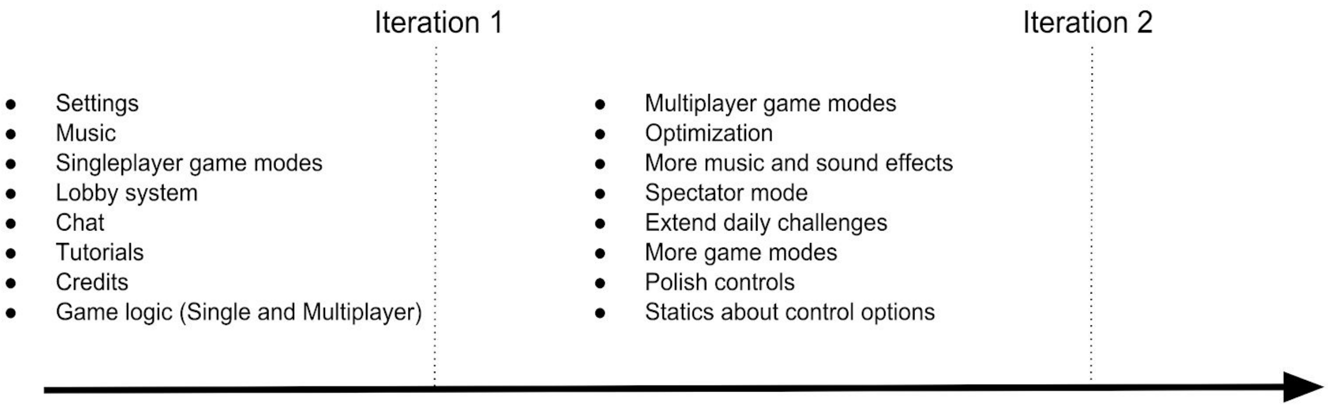


Figure 7.1 Milestones

We have decided about our milestones as indicated above. We have done everything that were supposed to be done until Iteration 1. Settings screen and functions are working correctly. We have covered some musics ourselves and added to the game. Singleplayer game modes (both logic and screens) are implemented and working properly. Tutorials of the implemented game modes are available in the game. In the credits screen, all of the developers names and a link to github are shown. Game logic is implemented for both singleplayer and multiplayer game modes and appears to be working properly. However, lobby and chat systems need to be optimized and extended. In the current version of the game, there are some bugs, e.g. it is possible to observe duplicates in the lobby list.

For Iteration 2, we are, in general, planning to do code optimization in terms of efficiency and readability and also add some more modes, i.e spectator modes (can be seen in Additional Functional Requirements). We also planning to store the statistics about the control choices of the players and finish times.

User Interface

Main Menu

This screen contains 6 buttons which will be used to navigate to other screens. Main Menu mockup can be seen in Figure 8.1.

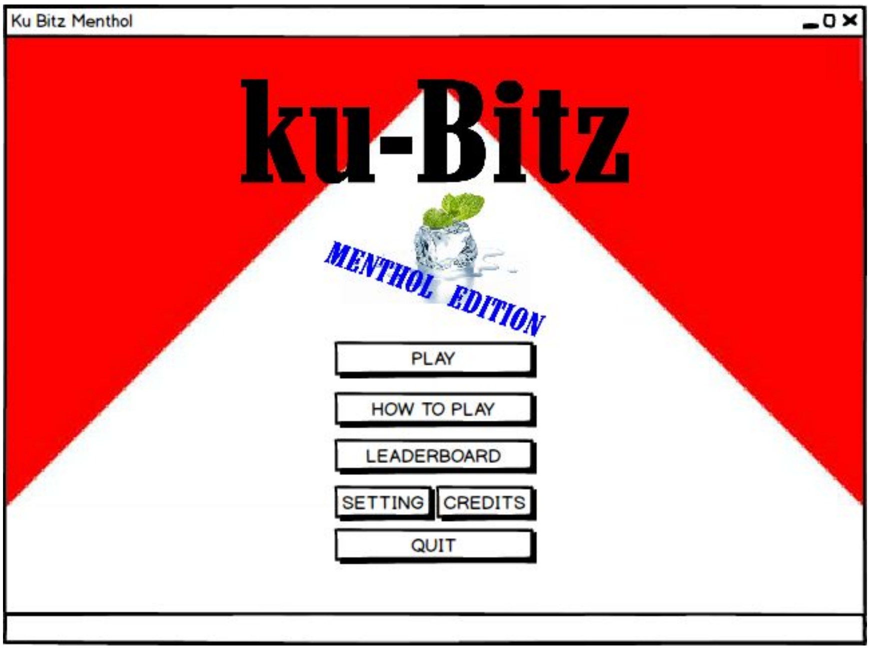


Figure 8.1 Main Menu Mockup

Play Screen

In this screen, player chooses the game mode that he/she wants to play. If the player chooses the Multiplayer option, he/she will be navigated to lobbies screen. If he/she chooses daily Challenge or Survival Mode buttons, the game will start with chosen game mode. Play Screen mockup can be seen in Figure 7.2.

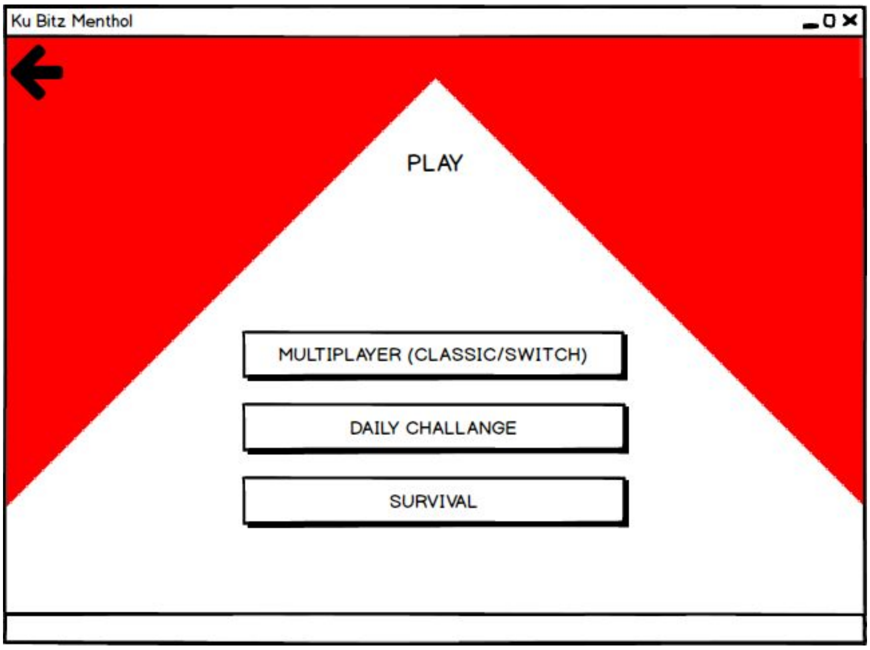
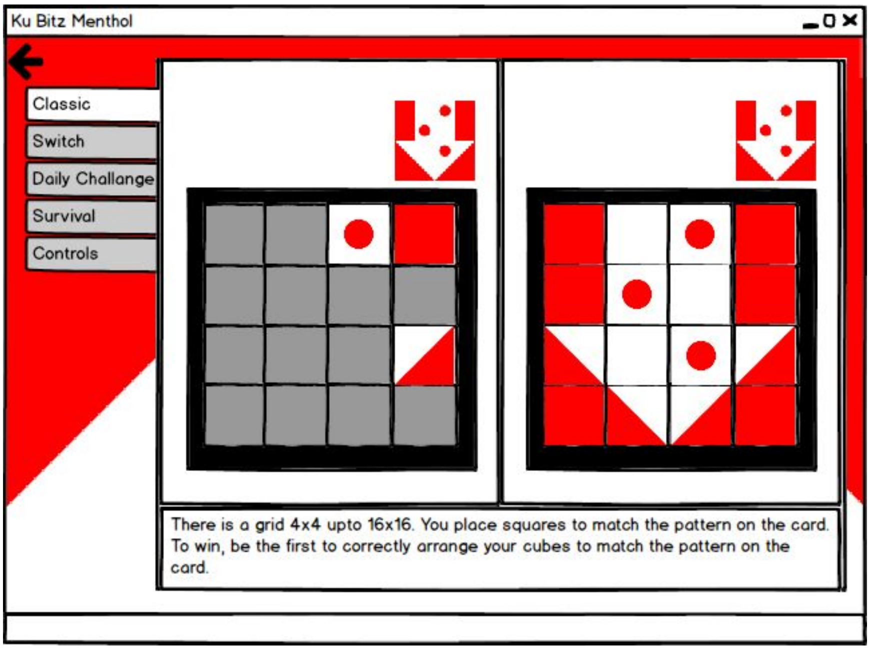


Figure 8.2 Play Screen Mockup

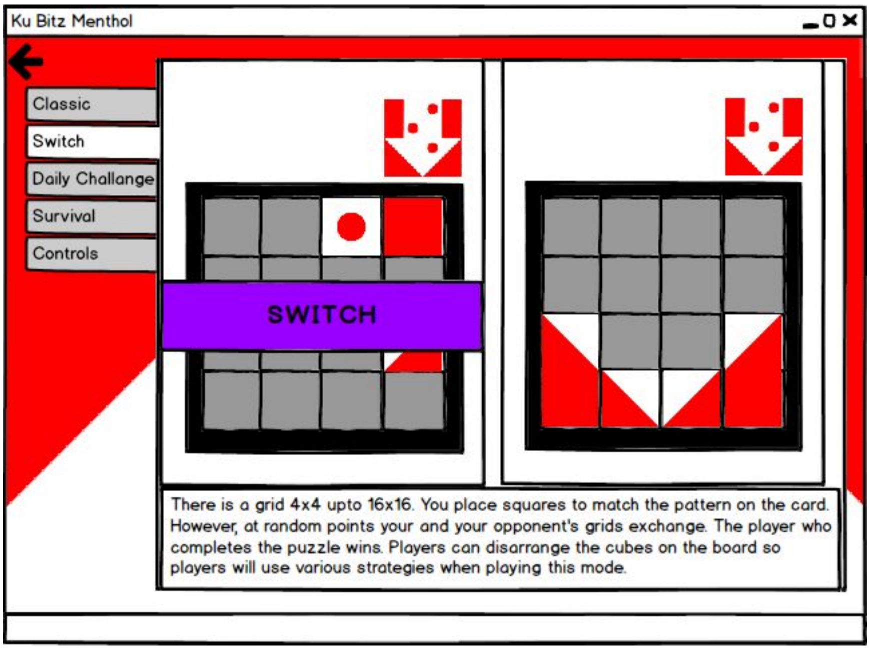
Tutorial Screen of Classic Mode



There is a grid 4x4 upto 16x16. You place squares to match the pattern on the card. To win, be the first to correctly arrange your cubes to match the pattern on the card.

Figure 8.3 Classic Mode Tutorial Mockup

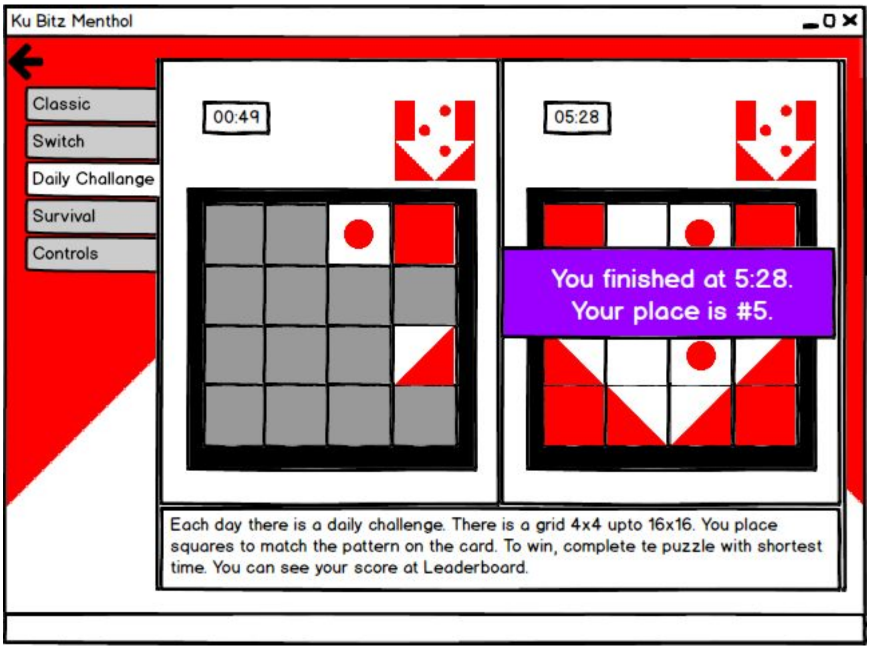
Tutorial Screen of the Switch Mode



There is a grid 4x4 upto 16x16. You place squares to match the pattern on the card. However, at random points your and your opponent's grids exchange. The player who completes the puzzle wins. Players can disarrange the cubes on the board so players will use various strategies when playing this mode.

Figure 8.4 Switch Mode Tutorial Mockup

Tutorial Screen of the Daily Challenge Mode

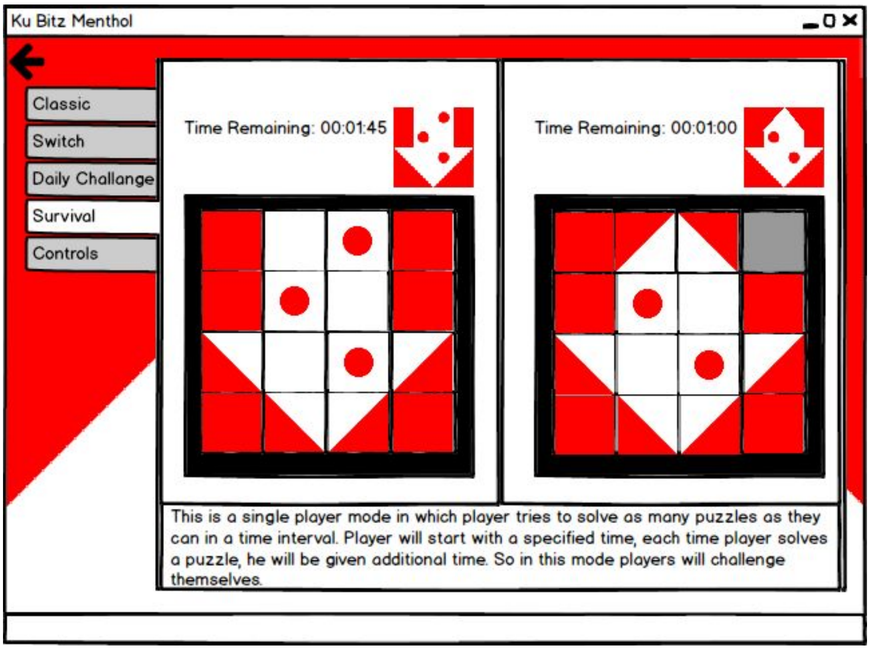


You finished at 5:28. Your place is #5.

Each day there is a daily challenge. There is a grid 4x4 upto 16x16. You place squares to match the pattern on the card. To win, complete te puzzle with shortest time. You can see your score at Leaderboard.

Figure 8.5 Daily Challenge Mode Tutorial Mockup

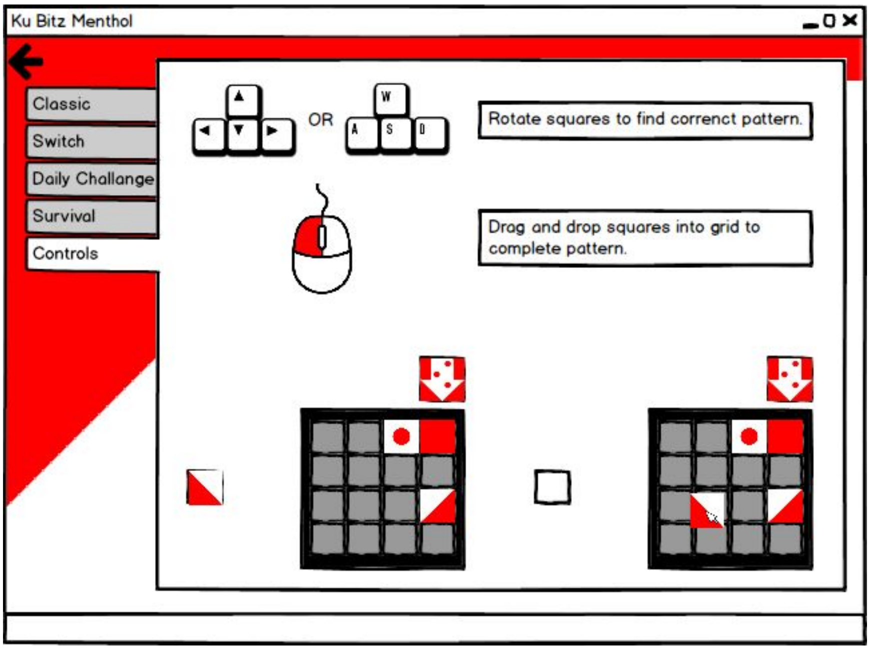
Tutorial Screen of the Survival Mode



This is a single player mode in which player tries to solve as many puzzles as they can in a time interval. Player will start with a specified time, each time player solves a puzzle, he will be given additional time. So in this mode players will challenge themselves.

Figure 8.6 Survival Mode Tutorial Mockup

Tutorial Screen of the Controls



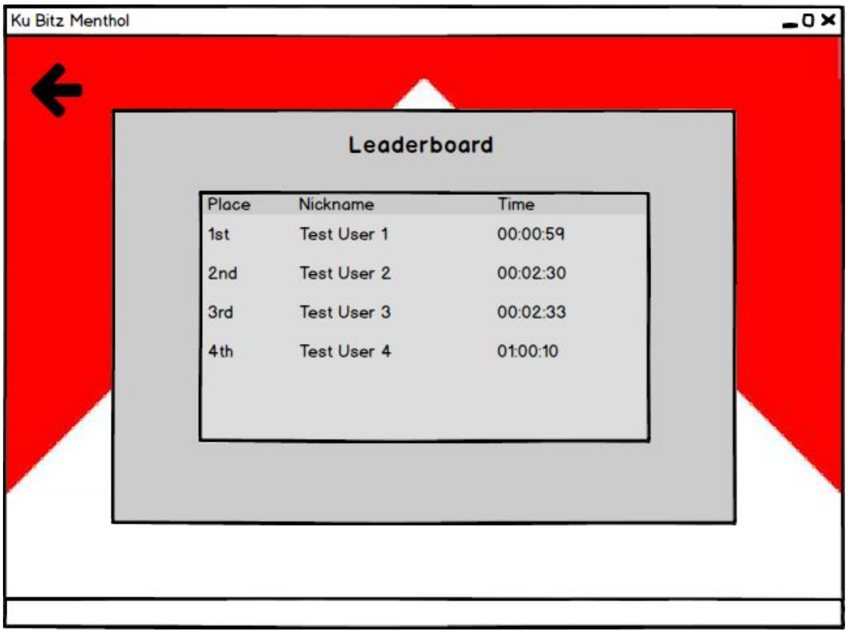
Rotate squares to find correnct pattern.

Drag and drop squares into grid to complete pattern.

Figure 8.7 Controls Tutorial Mockup

Leaderboard Screen

This screen shows the best scores of the day's challenge. Players' timings will automatically posted after playing Daily Challenge Mode. See Figure 8.7 for Leaderboard Screen mockup.

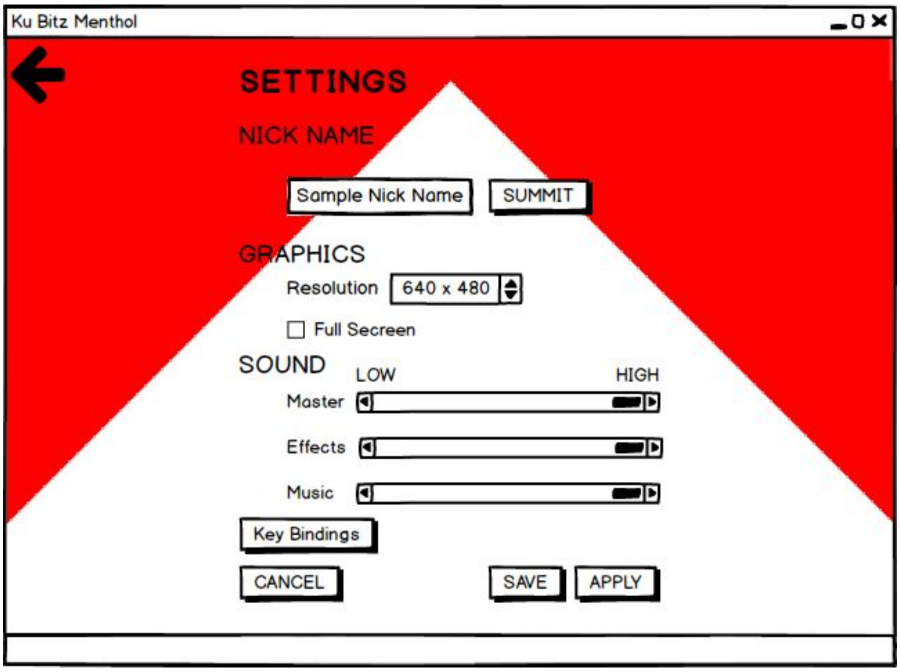


Place	Nickname	Time
1st	Test User 1	00:00:59
2nd	Test User 2	00:02:30
3rd	Test User 3	00:02:33
4th	Test User 4	01:00:10

Figure 8.7 Leaderboard Mockup

Settings Screen

This is the screen in which player can alter game settings. Player can change his/her nickname; Each nickname is unique thus player needs to submit his/her nickname to be checked whether it is unique or not. Player can also change the resolution or sound levels as can be seen in the Figure 8.8.



Sample Nick Name

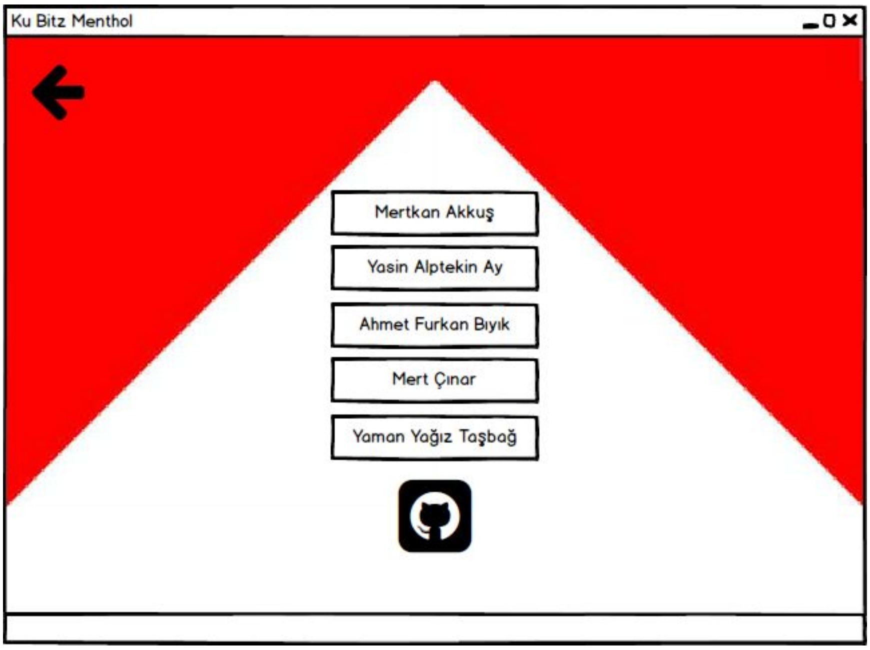
Resolution 640 x 480

□ Full Secreen

Figure 8.8 Settings Screen Mockup

Credits Screen

This screen shows the names of the people who worked on the project. Their names are displayed in a box. There is also a GitHUB button which will redirect to the project's repository page (See Figure 8.9).



Mertkan Akkuş

Yasin Alptekin Ay

Ahmet Furkan Bıyık

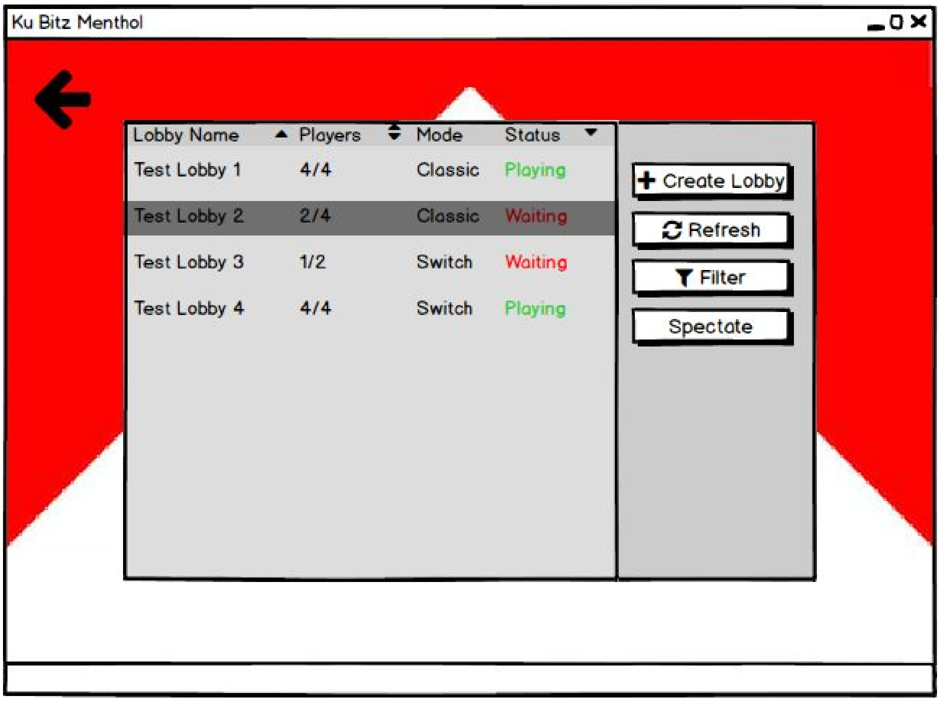
Mert Çınar

Yaman Yağız Taşbağ

Figure 8.9 Credits Screen Mockup

Lobby List Screen

After choosing multiplayer game modes from the play screen (see Figure 8.2) this screen will be shown to the player. In this screen currently available lobbies are shown to the player and player can join these lobbies by double clicking on them (assuming the lobby is neither full nor playing) which will direct him/her to the lobby screen (see Figure 8.13). Player can press on Create Lobby button which will redirect the player to the lobby creation screen (see Figure 8.12). Player can also refresh the page to see if any lobby is deleted or created. Player can also filter the lobbies (see Figure 8.11). A player can join a lobby as a spectator to watch the game by selecting the lobby and pressing the spectate button(see Spectator Mode). You can see the mockup for lobby list screen in the Figure 8.10.

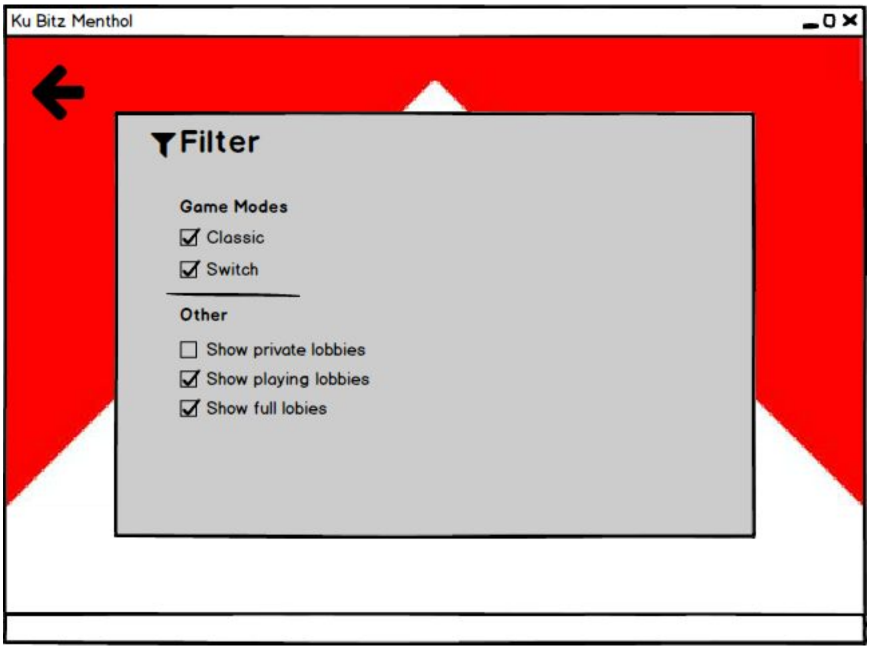


Lobby Name ▲	Players ⬍	Mode	Status ▼
Test Lobby 1	4/4	Classic	Playing
Test Lobby 2	2/4	Classic	Waiting
Test Lobby 3	1/2	Switch	Waiting
Test Lobby 4	4/4	Switch	Playing

Figure 8.10 Lobby List Screen Mockup

Lobby Filter Screen

In this screen the player can check checkboxes and save the filter settings to find the lobbies with desired properties (see Figure 8.11).



Game Modes

☑ Classic

☑ Switch

Other

□ Show private lobbies

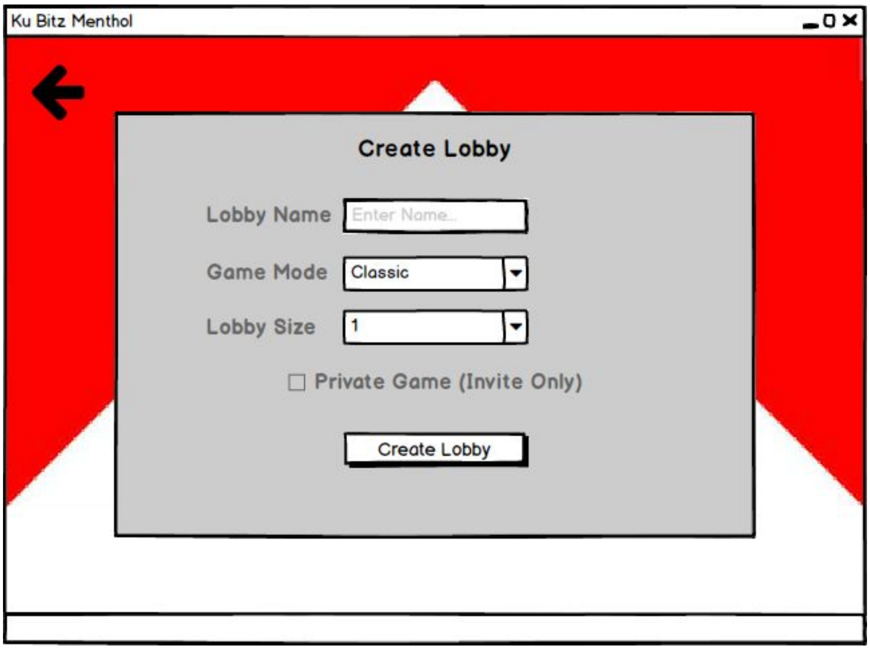
☑ Show playing lobbies

☑ Show full lobies

Figure 8.11 Lobby Filter Screen Mockup

Create Lobby Screen

In this screen the player will set the name, game mode and size of the lobby(see Figure 8.12). After creating the lobby, the player will be directed to the lobby screen(see Figure 8.13). The creator of the lobby will be the lobby admin who will be able to use some functions such as kicking a player (see Figure 8.13).



Lobby Name: Enter Name...

Game Mode: Classic

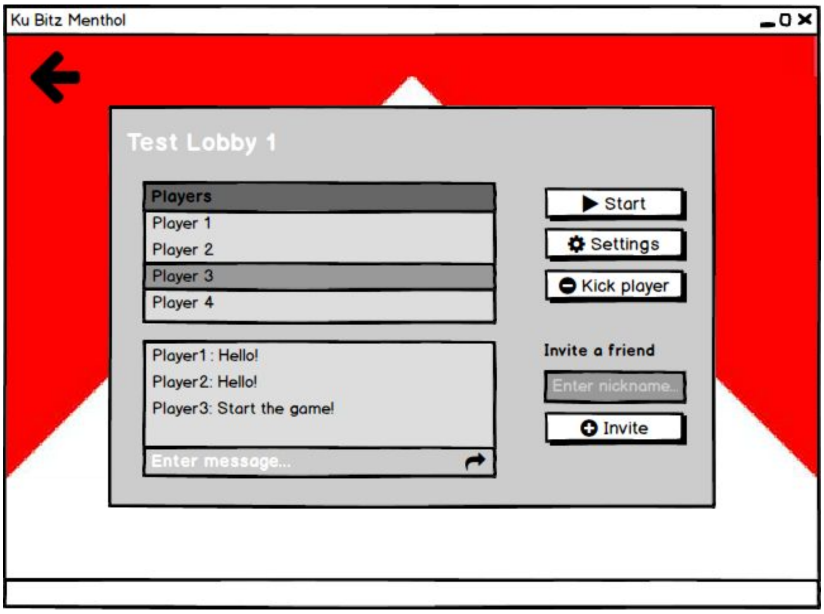
Lobby Size: 1

□ Private Game (Invite Only)

Figure 8.12 Create Lobby Screen Mockup

Lobby Screen

In lobby screen, the players that are currently in the lobby are shown in a list. The creator of the lobby is the lobby admin; Only the lobby admin can kick a player, change the lobby settings or start the game (see Figure 8.13). To kick a player, lobby admin should select the player on the list then press the kick button. Every player in the lobby can invite another player by typing the player's name to the field then pressing the invite button.



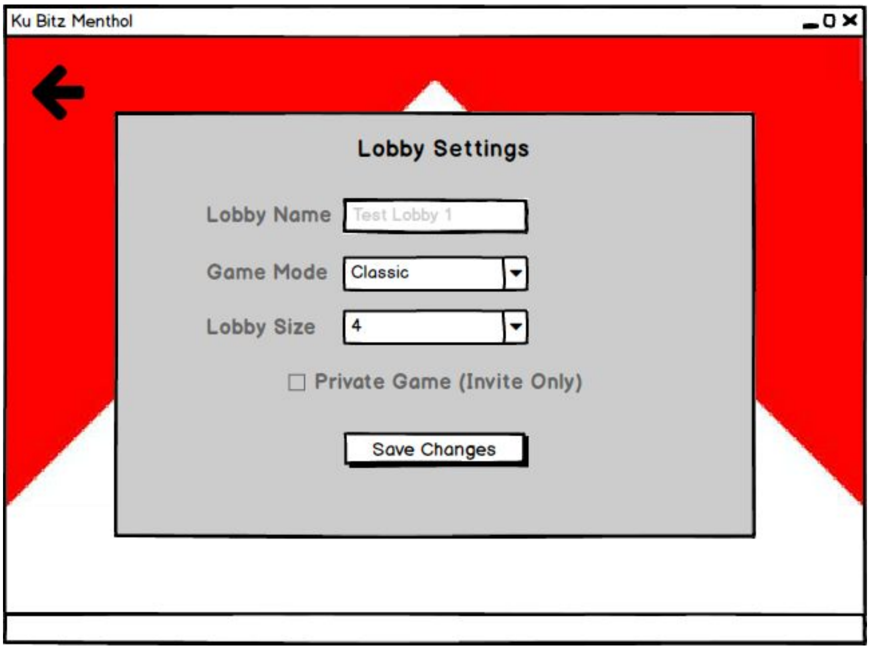
Players
Player 1
Player 2
Player 3
Player 4

Player1 : Hello!
Player2: Hello!
Player3: Start the game!

Figure 8.13 Lobby Screen Mockup

Lobby Settings Screen

In this screen the player (lobby admin), can change the settings of the lobby which were set in the lobby creation screen (see Figure 8.12).



Lobby Settings

Lobby Name: Test Lobby 1

Game Mode: Classic

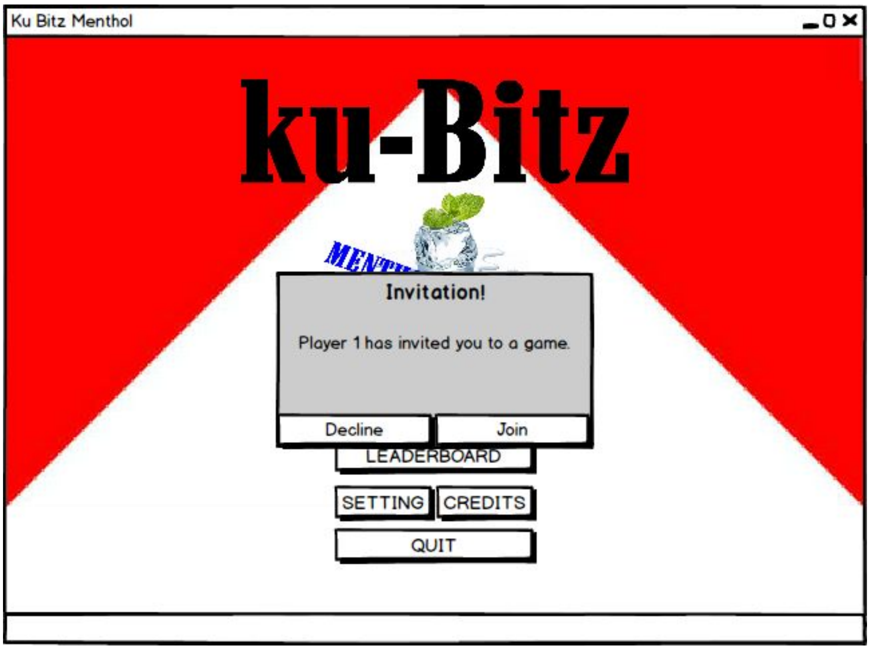
Lobby Size: 4

□ Private Game (Invite Only)

Figure 8.14 Lobby Settings Screen Mockup

Invitation Screen

When a player is invited to a game, a pop-up will be displayed regardless of the screen that is currently being display to the player. In the pop-up, name of the inviter is displayed with two options, decline and join. If player accepts the invitation, he/she will be directed to the lobby screen (see Figure 8.13), else the pop-up will disappear.



Invitation!

Player 1 has invited you to a game.

Decline | Join

Figure 8.15 Invitation Pop-up Mockup

Game Play Screen

This screen more or less the same for each game mode, in classic and switch mode time is not displayed. The selected cube displayed on the left. The selected side of the cube is displayed brighter, on the each side of the cube, the next side is displayed darker. Player can rotate the cube by clicking on the darker sides of the cube or using W A S D buttons. Player can also rotate the cube clockwise and counterclockwise with Q and E buttons. Placement is done by clicking the related cell of the grid board(see Figure 8.16).

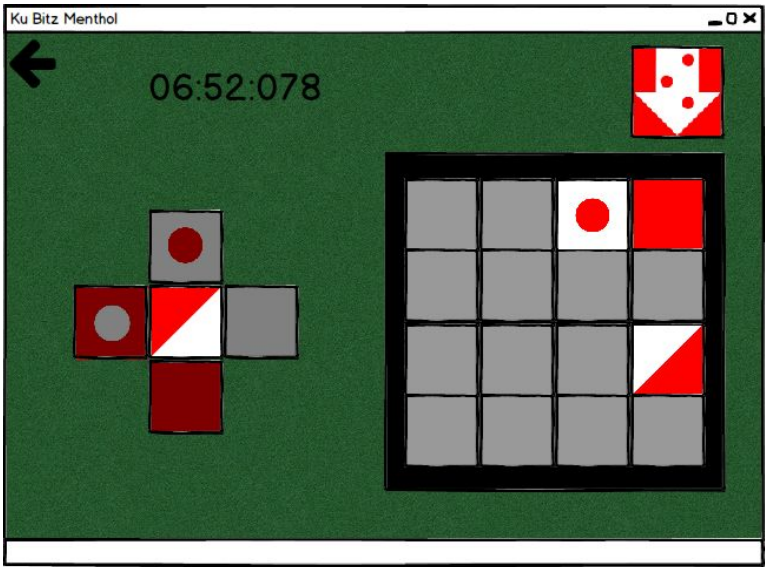
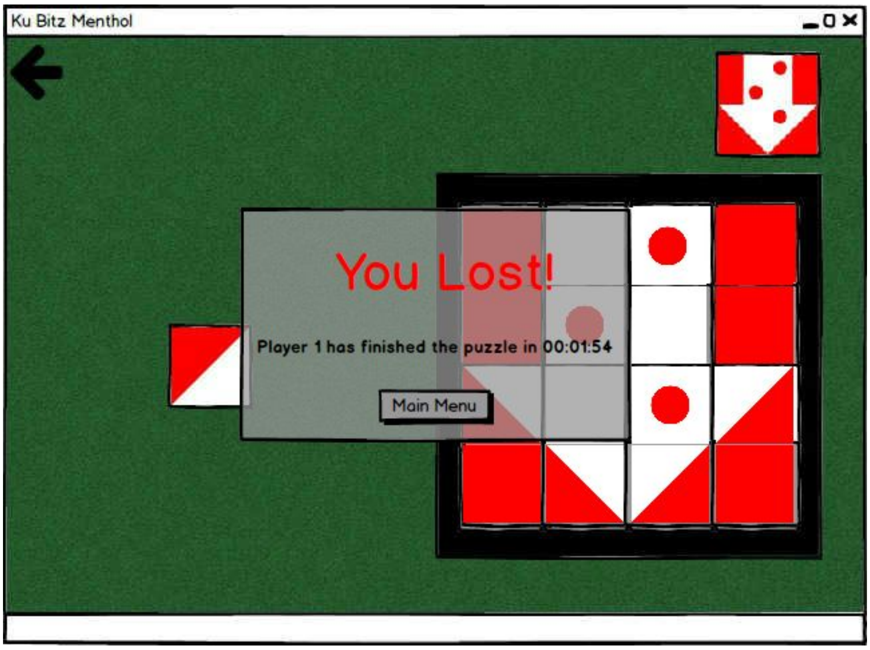


Figure 8.16 Game Play Screen Mockup

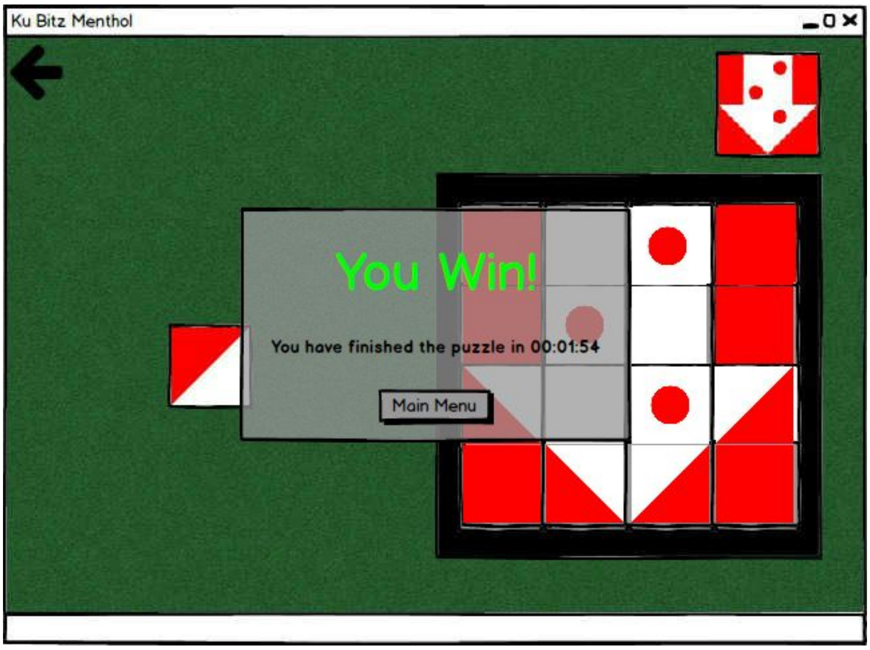
Win/Loss Screen



You Lost!

Player 1 has finished the puzzle in 00:01:54

Figure 8.17 Game Lost Pop-Up Mockup



You Win!

You have finished the puzzle in 00:01:54

Figure 8.18 Game Won Pop-Up Mockup

Spectator Screen

The spectator function is explained in the previous part (see Spectator Mode). After game starts, this screen will be shown to the spectators of the lobby, spectators can see the progress of all the players in the game.

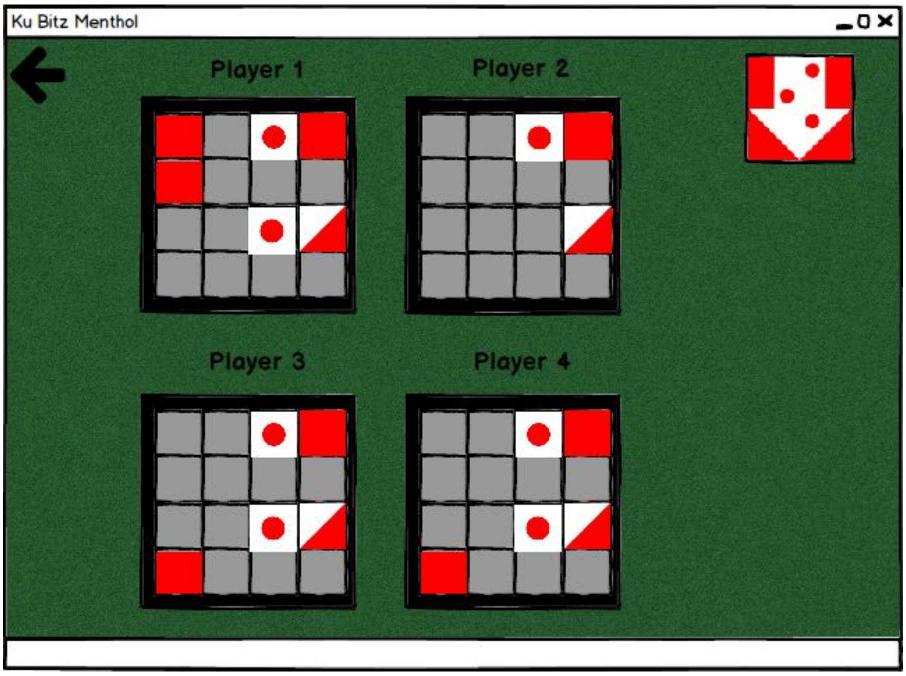
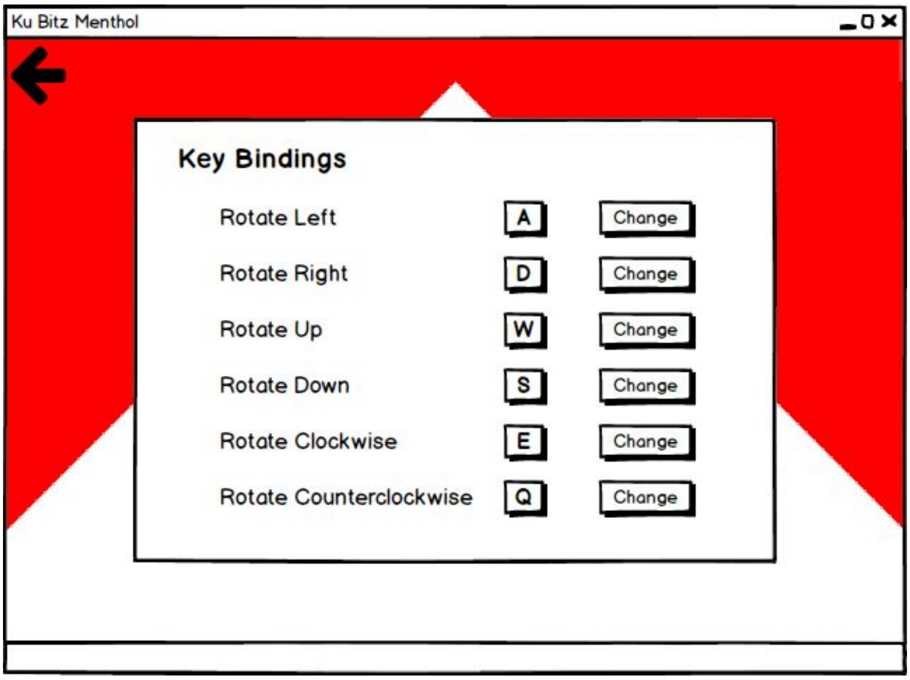


Figure 8.19 Spectator Screen

Key Bindings Screen

In this screen players can change the key bindings that are required to rotate the cube. After pressing the change button which next to each rotation function, the player should press the key they want to bind the function to.



Key Bindings

Rotate Left	A	Change
Rotate Right	D	Change
Rotate Up	W	Change
Rotate Down	S	Change
Rotate Clockwise	E	Change
Rotate Counterclockwise	Q	Change

Figure 8.20 Key Bindings Screen

References

1) https://www.mindware.orientaltrading.com/q-bitz-a2-44002.fltr

2) http://sburngdl.weebly.com/uploads/8/8/9/0/88900840/arrow_orig.png

3) http://pngimg.com/uploads/cursor/cursor_PNG91.png

4) https://www.kisspng.com/png-ice-mint-melting-menthol-wallpaper-ice-522060/

5) https://image.freepik.com/free-icon/mouse-ios-7-symbol_318-38753.jpg