Object-Oriented Software Engineering

CS319

Term Project
Design 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

Contents

1. Introduction

1.1 Purpose of the System

Q-Bitz is originally a boardgame where players push the limits of speed and memory in different rounds of the game. Our purpose is to create a desktop version of this game with more fun by adding more futures. We will implement all the features of the real game to our project. However, our game will have additional modern features such as the game modes. We will offer the players a smooth multiplayer experience. Our game has singleplayer mode which is not available in the board game. We aim our desktop version of Q-Bitz to be adjustable by the users, so they can adapt in less than 10 minutes. Our game will also offer a pleasurable visual and audial environment for the players.

1.2 Design Goals

1.2.1 User Friendly Interface and Ease of Controls

One of the main goals of our user interface design is to make the game easier to play. Since Q-Bitz is a game designed for children, menu navigation should be easy to understand. A player should not spend much time on finding his/her friend's lobby. Game controls should be designed in a way that gameplay should be easy. Players should play as fast as they can to win, so UI design and game control decisions should not be obstacles for a player.

1.2.2 Performance

A smooth gameplay experience is one of our design goals. Some of the game modes of the game require players to compete against time, so poor performance should not slow down players.

1.2.3 Entertainment and Attractiveness

Being a game for children, Q-Bitz fails on attracting adults and presenting them a proper entertainment. Software implementation of the game should have additional game modes to extend the competitiveness between players.

Players can communicate each other easily in a board game, a software implementation that does not provide its players the ability to communicate with their opponents would be unappealing. The software version should have a chat system in game lobby so players can chat with each other.

1.2.4 Why Java?

- Practicality
- Backwards compatibility
- Scalability/Performance/Reliability
- Freshness

1.2.5 Why Spring?

- Exposing RESTful services
- Dependency injection approach
- Powerful database transaction management capabilities
- Integration with other Java frameworks

1.2.6 Why Maven?

- Easy build process
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

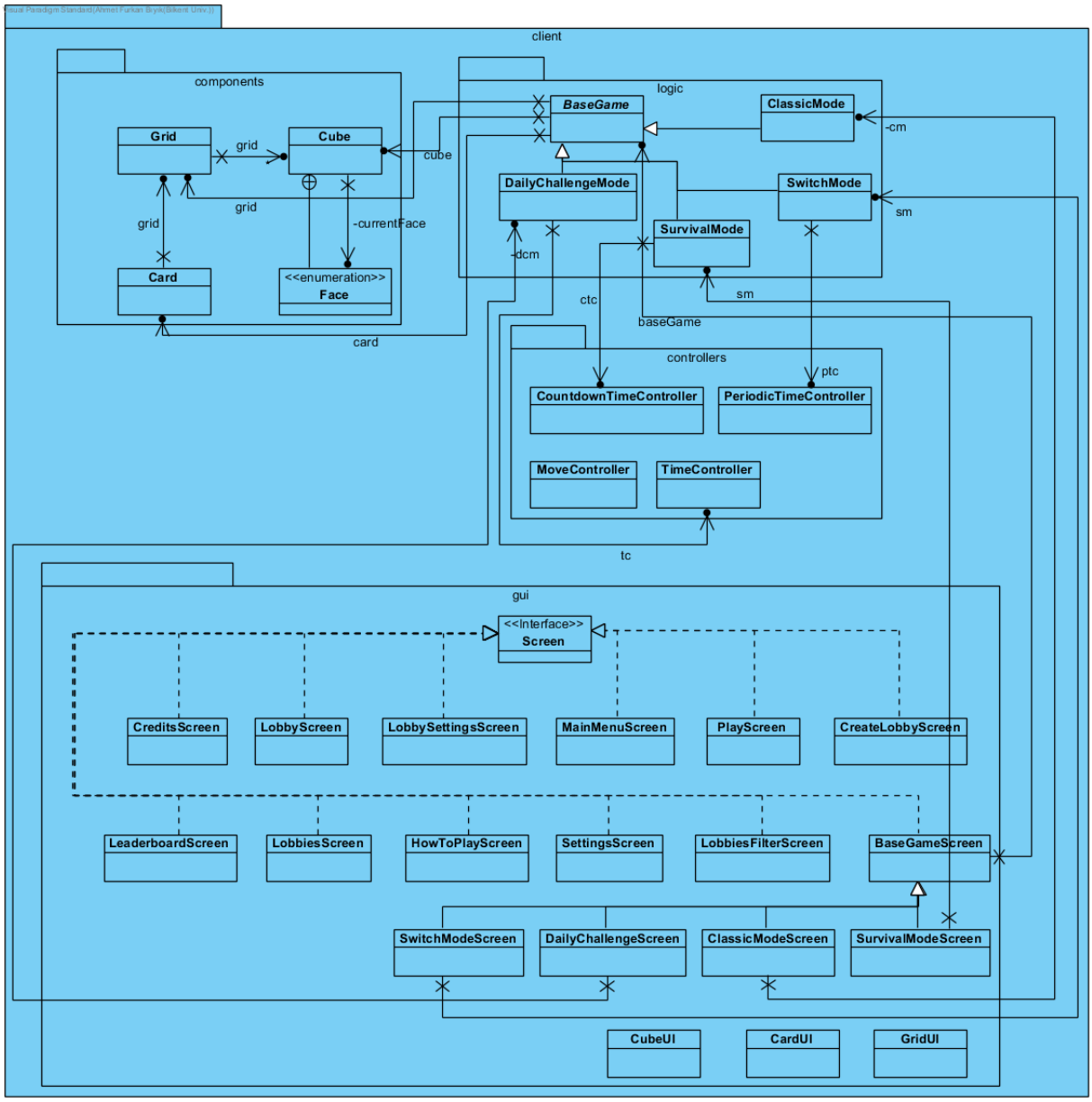
(https://maven.apache.org/what-is-maven.html)

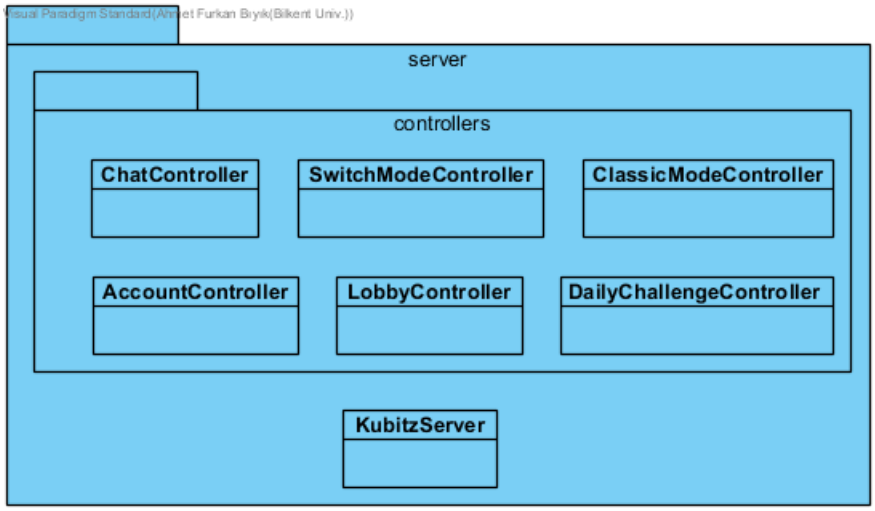
2. High-level Software Architecture

2.1. Overview

Main design issues and decisions will be discussed in this report. And our proposed design for the system will be illustrated. The overall architecture of the system is described and in the following sections the subsystems are detailed. Various diagrams and figures are used wherever they enhance the understanding of the system.

2.2 Subsystem Decomposition





Our game can be decomposed into two main parts: client and server. Client part consists of 4 packages: gui, components, logic and controllers. GUI package includes the classes that are responsible of creating graphical user interfaces for all screens within the game. Controller package has the classes that are responsible of time and move controls of some game modes. Logic package includes the classes that implement game functionalities. Components package includes object classes needed for the game like cube, card and grid (game board).

Server part has endpoints for each controller class in client such as Chat Controller, SwitchModeController, ClassicModeController, AccountController, LobbyController and DailyChallenge controller. It also includes a KubitzServer which implements communication with client.

2.3 Hardware Software Mapping

This is a game project so there is no complex hardware usage. Following hardware components are mapped to our game using Java API:

- Keyboard: For cube rotation manipulation.
- Mouse: For menu navigation, also for cube rotation and placement.

- Monitor: The GUI is displayed on monitor.
- Speakers: For sound effects and game music.

2.4 Persistent Data Management

In our game, we do not have much data management issues however there are some points which we want to have persistent data management to create a seamless multiplayer experience.

- Leaderboard
- Handling connectivity problems: We don’t want players to experience connection issues like losing the connection to the party and leave during the game.

2.5 Access Control and Security

Since our game does not store any important data, we currently don’t have security concerns.

2.6 Boundary Conditions

- When a player disconnects during the game, other players in the same lobby are informed that the player has disconnected.
- Every data related to multiplayer game modes are stored in the server.

3 Low Level Design

3.1 Object Design Trade-offs

3.1.1 Performance versus Response Time

Response time is one of most important factor since our system provides smooth gameplay experience while compete against players and time. Therefore, more performance required to decrease response time.

3.1.2 Efficiency versus Portability

Our game focuses on porting between different environments easily. With Java portability can be achieved easily. However, efficiency decreases since Java communicates machine via virtual machine.

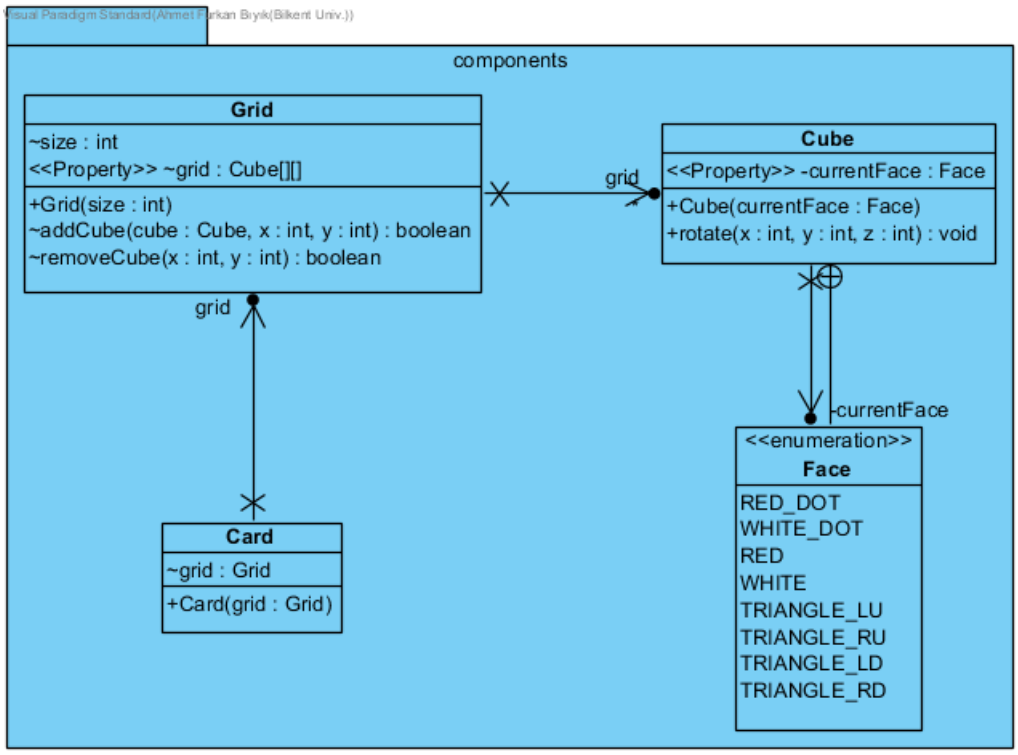
3.2 Packages and Final Object Design

Our project consists of two sub-projects which are client and server sides. Both packages will be explained further in their sub-sections.

3.2.1 Client Package

This package consists of four packages which are components package, logic package, controllers package and gui package. These packages will be explained below.

3.2.1.1 Components Package



This Package contains the main components of this game, such as the parts in the original board game Q-bitz which are the simply game cards, cubes and a grid.

These main components are basically used like; Players/player check the figure on the game card and try to arrange the cubes on the grid as same as the figure.

3.2.1.1.1 Grid

In Kubitz game, we have a class called Grid which is the base where the cubes will be placed on. It is designed as 4x4, so 16 cubes will be fitted when the grid is full.

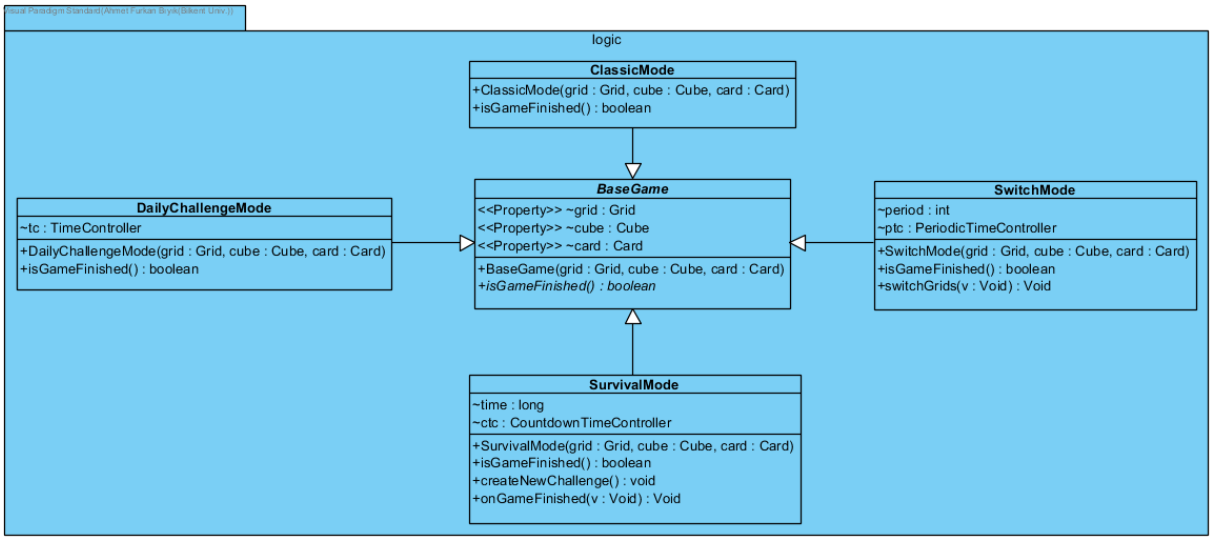
3.2.1.1.2 Cube

Cubes are used by the players to obtain the same figure as the game card. A cube has 6 faces and all these faces contain different shapes on a single cube. However, all the cubes are identical in the game. In the gameplay there is one particular cube (which is also same with the other cubes) that will be shown to the players, and be moved by the players to the grid.

3.2.1.1.3 Card

Card class is used in all of the game modes such as the other components. Every card has a different figure on it and the players will try to arrange the cubes according to these cards. The cards will change every round depending on a game mode in a game but it is possible for a player to see a card twice or more by chance.

3.2.1.2 Logic Package



This package mainly contains the game modes which are classical, survival, switch and daily challenge. There is also a class called BaseGame which will be mentioned below.

3.2.1.2.1 BaseGame

There is an abstract class called BaseGame in this package, which is the parent of the other game modes. This class contains the methods that is common with all the other game modes.

3.2.1.2.2 SurvivalMode

Survival Mode is a game mode where players are competing against time. A player encounters with game cards one after another if they manage to keep solving. Every game card they manage to finish will give them extra time. This mode is a singleplayer game mode.

3.2.1.2.3 DailyChallangeMode

Daily Challenge game mode is a mode that is created by the developers every day for the players to solve. Every player has one shot to try this mode every day. Players will come upon several game cards and they will be listed in the leaderboard according to the total time that they finish all these puzzles. This mode is a singleplayer game mode.

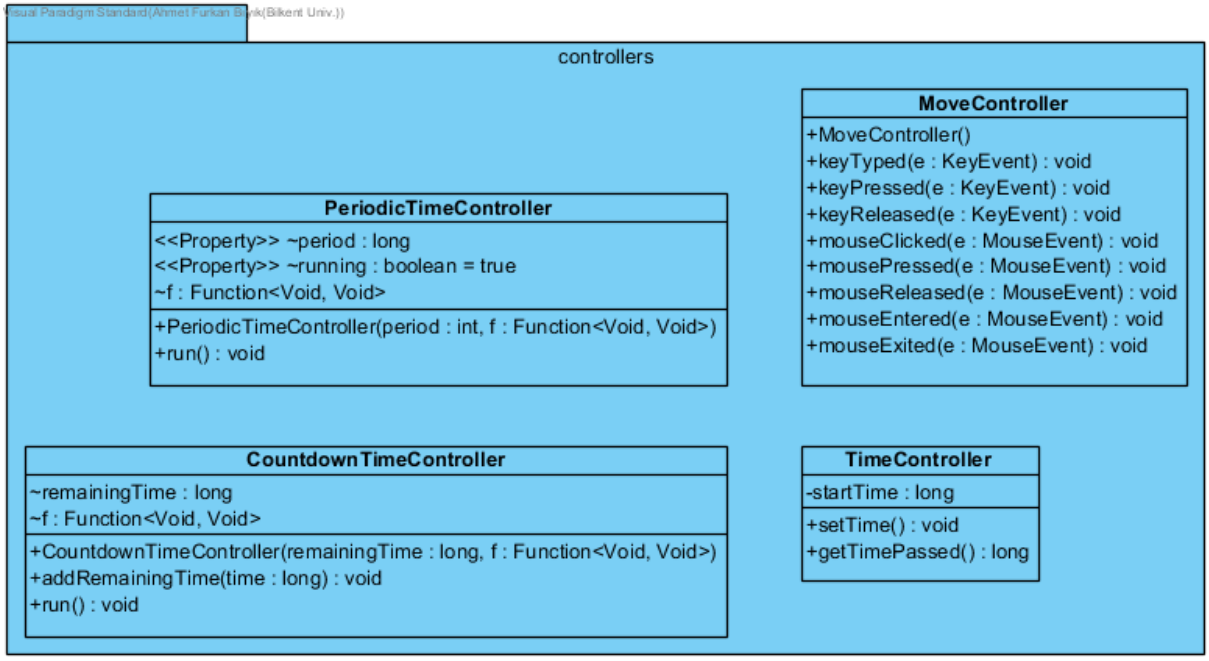
3.2.1.2.4 ClassicMode

Classic Mode is a multiplayer mode. The players can take part in this game by either creating their own lobby or joining an existing lobby from the lobby list. This mode is the basic mode where players try to finish the game first.

3.2.1.2.5 SwitchMode

Switch mode is also a multiplayer mode where the players can take part by either creating their own lobby or joining an existing lobby from the lobby list. Two players will try to finish the game first. In every 15 seconds, they players will switch the boards. If the overall placement of the board is correct, the game is over, and the winner will be displayed.

3.2.1.3 Controllers Package



This package consists of the main controllers for the game that will be described below in more detail.

3.2.1.3.1 CountdownTimeController

This time controller class is created to keep track of the time in the survival game mode. It has the time methods which are used in the survival mode. Player starts the game with an initial time which begins to decrease when the game starts and the player gains extra time when he/she solves a game card.

3.2.1.3.2 MoveController

This class is a controller to track the user actions such as mouse and keyboard activities.

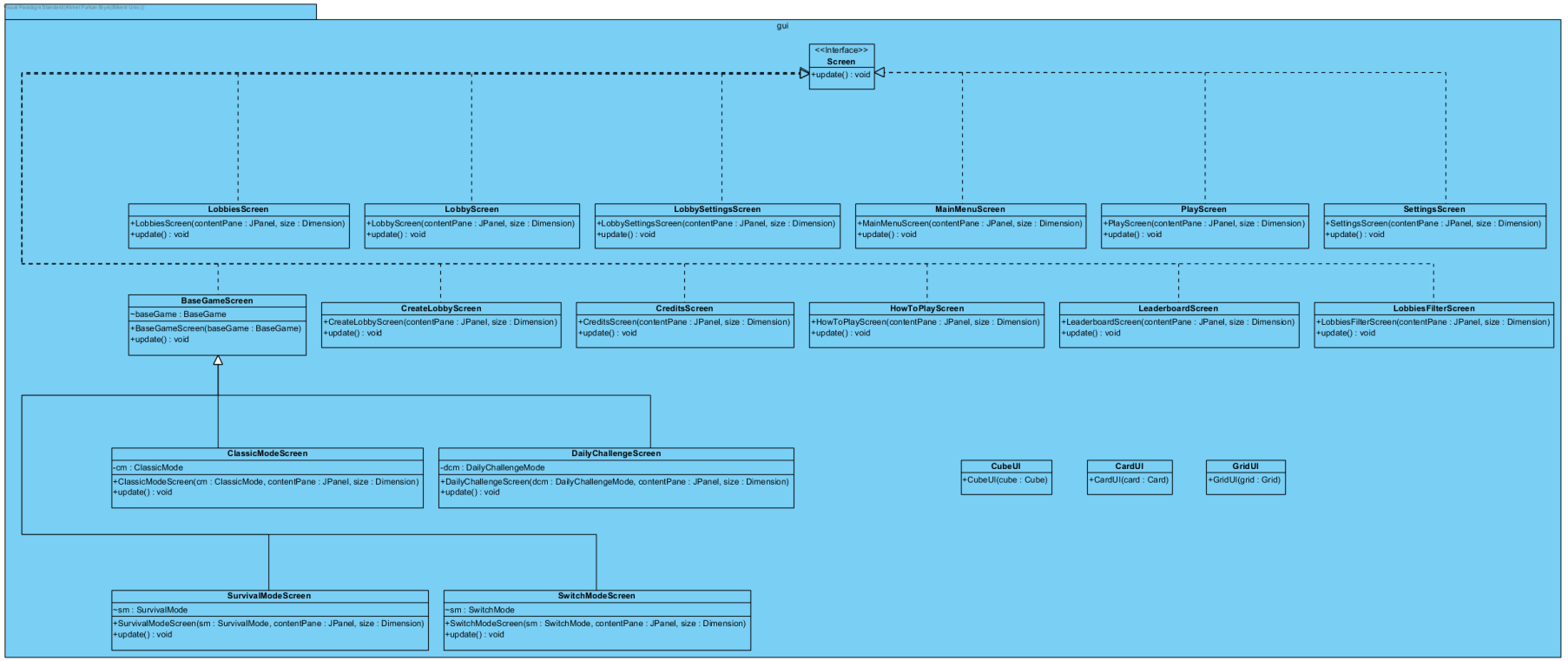
3.2.1.3.3 PeriodicTimeController

In the Switch game mode, the game must keep track of the time in predefined time periods. This time controller keeps these periods and tells when to switch the boards.

3.2.1.3.4 TimeController

For the Daily Challenge Mode, total time passed to solve all the game cards must be saved by the game. Therefore, time controller saves this time to make the player listed in the leaderboard according to the finish time.

3.2.1.4 Gui Package



This package contains user interfaces and screens which are credits, lobby, lobby settings, create lobby, lobbies, lobbies filter, main menu, play, leaderboard, how to play, settings, base game, switch mode, daily challenge mode, classic mode and survival mode screens, and cube, card and grid interfaces. There is also an interface class named screen which will be mentioned below.

3.2.1.4.1 Screen

This class is an interface which is a basis for other screens. It declares the common functions and properties.

3.2.1.4.2 CreditsScreen

This class is not a mostly interactive screen. It shows the credits of the game.

3.2.1.4.3 LobbyScreen

This class connects server and communicates server all the time. It shows players in lobby and there is a friend invitation interface. There is also a chat box in which the players in the lobby can communicate with each other. In addition, it has some interactions for lobby leader such as start game, kick player, settings.

3.2.1.4.4 LobbySettingsScreen

This class shows the settings of the lobby which only lobby leader can access. In this class lobby name, game mode, lobby size and status of the game lobby private or not can be set. This class also communicates with server.

3.2.1.4.5 CreateLobbyScreen

This class shows the options of the lobby which will be created. In this class lobby name, game mode, lobby size and status of the game lobby private or not can be set. Then it creates a lobby and communicates with server.

3.2.1.4.6 LobbiesScreen

This class shows the list of the lobbies which are created that are gotten from the server. It gets the list of joinable lobbies from the server by the means of sending the current filter preferences to the server and refreshes the list. When a lobby is double clicked on the list, it shows the lobby screen. Once the create lobby button pressed, it shows create lobby screen.

3.2.1.4.7 LobbiesFilterScreen

This class has options for the games showed in the lobbies screen. It updates the current filter preferences.

3.2.1.4.8 MainMenuScreen

The current screen consists of buttons which have the responsibilities to show the respective screens.

3.2.1.4.9 PlayScreen

This class has buttons that are shows respective game choices such as multiplayer, daily challenge or survival. When multiplayer is clicked, it shows lobbies screen.

3.2.1.4.10 LeaderboardScreen

This class shows list of players with their scores in the current daily challenge. It fetches the data from the server.

3.2.1.4.11 HowToPlayScreen

This class shows the tutorials of the game. There are few pages that player can go between information.

3.2.1.4.12 SettingsScreen

This class nick name can be changed by sending request to server. There are options for graphics and sound. Settings can be save, apply or cancel.

3.2.1.4.13 BaseGameScreen

This class has common interfaces for other game modes. It has base game logic as a property. It renders the grid, the card and the cube that will be placed.

3.2.1.4.14 SwitchModeScreen

This class extends base game screen. It has a switch mode logic property. It shows time left from switch mode logic.

3.2.1.4.15 DailyChallangeModeScreen

This class extends base game screen. It has daily challenge mode logic property. It shows time passed from the daily challenge mode logic.

3.2.1.4.16 ClassicModeScreen

This class extends base game screen. It has a classic mode logic property.

3.2.1.4.17 SurvicalModeScreen

This class extends base game screen. It has a survival mode logic property. It shows time left from the survival mode logic.

3.2.1.4.18 CubeIU

This class extends JPanel and will be organized to produce our custom cube panel. Its constructor takes a Cube instance, which is the logic part, as a parameter.

3.2.1.4.19 CardUI

This class extends JPanel and will be organized to produce our custom card panel. Its constructor takes a Card instance, which is the logic part, as a parameter.

3.2.1.4.20 GridUI

This class extends JPanel and will be organized to produce our custom grid panel. Its constructor takes a Grid instance, which is the logic part, as a parameter.

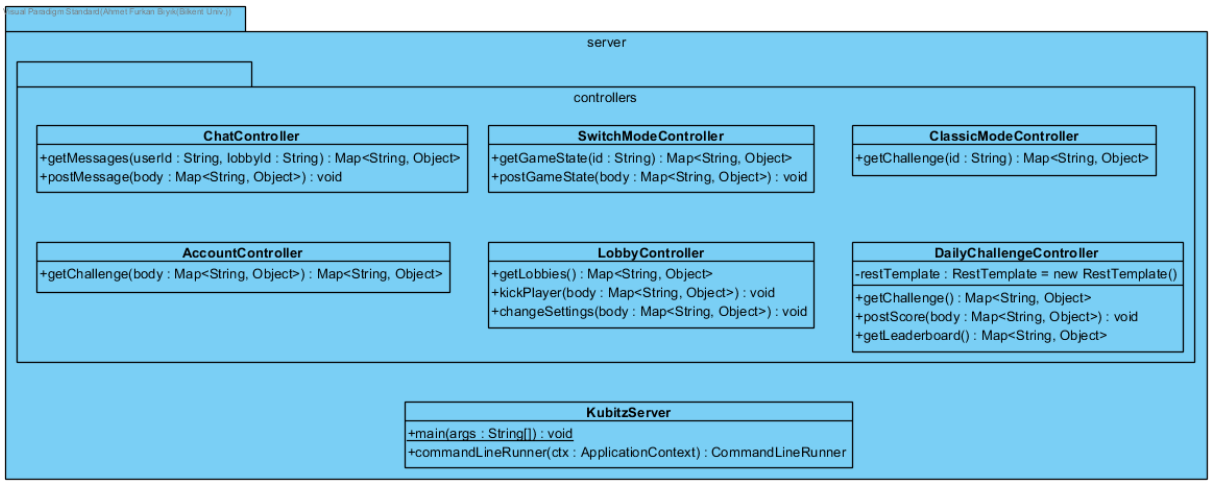
3.2.2 Server Package

We have used Spring framework of Java to implement the server. The server configurations are specified in /src/main/resources/application.properties.

3.2.2.1 KubitzServer

This is the main class which is obliged to start the server. It fetches the configurations from application.properties, which we have mentioned earlier.

3.2.2.2 Controllers Package



In this package, there are REST controller classes which are meant to be handling the communication between the players (clients).

3.2.2.2.1 ChatController

There are two APIs in the ChatController. It controls the chatting feature in a lobby. One of the APIs is for posting a message and the other one is for fetching messages.

3.2.2.2.2 SwitchModeController

SwitchModeController is for the management of the game mode Switch. There are two APIs: One is for fetching the game state of the opponent and the other is for posting the game state of the player.

3.2.2.2.3 ClassicModeController

There is only one API in ClassicalModeController which provides the same challenge to all players in the lobby.

3.2.2.2.4 DailyChallangeController

DailyChallengeController is for the management of the game mode Daily Challenge. There are three APIs: One is for getting leaderboard, the other is for posting the score of a player and the last one is for getting the challenge of the day.

3.2.2.2.5 LobbyController

In the LobbyController, there are three APIs. One is for fetching lobbies, the other is for kicking player from the lobby and the last one is for changing the settings of the lobby.

3.2.2.2.6 AccountController

It has the API that provides login and sign up feature for a player.

4 References

https://www.azul.com/4-reasons-java-still-1/

http://springtutorials.com/twelve-reasons-to-use-spring-framework/