CS 319

Object-Oriented Software Engineering Analysis Report

Backgammon

Group 1-D

Mert Armağan Sarı

Doğukan Altay

Berkalp Yılmaz

Ömer Sakarya

Table of Contents

1. Introduction

We decided to design and implement the Backgammon which is a classic board game exists for over 5,000 years. It is played by two persons and the objective is to remove all of one's own chips before the opposite does. It is basicly one of the most common strategy games played with dice. The game we plan to build, will be a dektop application which runs on Windows operating systems and controlled with mouse.

This report describes the game's rules and gameplays. Functional and non-functional requirements with use case models are also included in this paper.

2. Overview

2.1. Gameplay

Backgammon is a classic two player board game. Each player starts with 15 checkers placed on the board, and the player's goal is to remove his/her pieces before his/her opponent removes them. At the beginning of the game, each player rolls one die, and the one who rolls the higher value goes first. The player then moves his/her checkers forward by the total number on both dice. For example, rolling a 3 and a 6 lets the player move one piece forward by 3 spaces, and then move either the same piece or a different piece forward 6 spaces.

2.2. Blocking and Hitting

The players can move their pieces to blank spots, spots with their pieces already on them, or spots that have at most 1 opponent piece on them. If a player moves to a spot with 1 opponent piece on it, that piece is "hit" and is removed to the "bar".

On the next turn, that player is then forced to enter pieces from the bar back onto the starting area of the board before being allowed to make any other moves. If a space has 2 or more of a player's pieces on it, the opponent cannot move to that space and that player's pieces are "safe".

2.3. Bearing Off

Bearing of is the phase when the players start to collect their pieces from the board. It can only happen once all of a player's pieces are in the final 6 spaces of the board. Players can bear off any piece using an exact dice roll, ex. use a 3 roll to bear off a piece that is 3 away from the end. However, a roll may not be used to bear off checkers from the lowered number spot unless there are no pieces in higher spots.

2.4. Gammon

If one player has not gathered all of the pieces in his/her base space and the other player has finished bearing off his/her pieces on the board, then the player has lost a *gammon*, which counts for 2 round points.

2.5. Checker Colors

The players have the chance to change their checkers' colors during the game. They can pick any color from 3 defined colors in this game. The samples for these three colors are shown below:

3. Requirement Specification

3.1. Functional Requirements

3.1.1. Play Game

Backgammon is a board game which is implemented in Java. The main purpose of the game is collecting all of pieces before opponent does. Before starting the game, players should determine how many rounds they will be play and decide their nicknames. To choose who will start first, both players roll a die and the player with the bigger roll starts first. The round finishes when a player collects all of his/her pieces. To win the game players should win one more than half rounds of their choice.

During the game, players are also able to hit each other’s checkers to slow each other down and make their checkers “safe” to prevent their opponent’s movement. These rules spices up the game and forces players to create strategies and prevents the game being a roll’n luck game.

3.1.2. Change Settings

Players have opportunity to change some settings for game functionalities and cosmetic changes. The settings that can be changed by players are:

- Sound On/Off
- Change Checker Color

Players can turn on or off the game sounds from Settings menu. In addition, players can change the color scheme of their checkers to add more customization to the game. As default, the color scheme is Black/White but they have options to use Red/Black, Red/White.

3.1.3. How-to-Play

To keep players aware of the game rules. This game offers some pictures with explanations for players to let them understand the basics clearly.

Those explanations are about:

- Rules of Backgammon and the aim of the game.
- Player controls
- Some example moves with related pictures

Since Backgammon is a game that demands decent strategies to beat opponents, knowing the rules will enhance the experience for the players. Pictures help the players to understand the game more precise and allow them to create basic strategies.

3.1.4. Pause Game

The game can be paused by players while the game is on and continue wherever they left. When game is paused players have a chance to access some of the settings. Changeable setting(s) during the Pause screen are:

- Sound On/Off

If the player close the game while waiting in Pause screen, all of the progression within the game will be lost. There is no chance to save the game for playing afterwards.

3.1.5. View Credits

Players can reach the information about the game developers and some of their contact information. With this information, players can help developers to fix bugs in the game or make suggestions to improve it.

3.2. Non-Functional Requirements

3.2.1. Game Performance

As the developers of Backgammon game, we are designing the program with high performance with low system requirements. Since the game does not consist of complex animations and displays, we are focusing on lowering the needs as much as possible and eliminating the bugs related to the diffrence in systems.

3.2.2. Graphical User Interface

We are planning to make a user friendly UI that is detailed as much as needed and easy to use. By keeping the design neat, users will be more comfortable and make less mistakes during the game. We want to make Backgammon Game UI simple yet comprehensive.

3.2.3. Extendibility

Our motivation is that making a software design that we can be able to upgrade and improve. The Backgammon Game will be able to renew itself for future technologies and ideas. In case of willing to add some new rules to the game is very important for our design. Backgammon Game will be designed according to ease these needs.

4. System Model

4.1. Use Case Model

The use case model of Backgammon game is shown below with some detailed sample use cases:

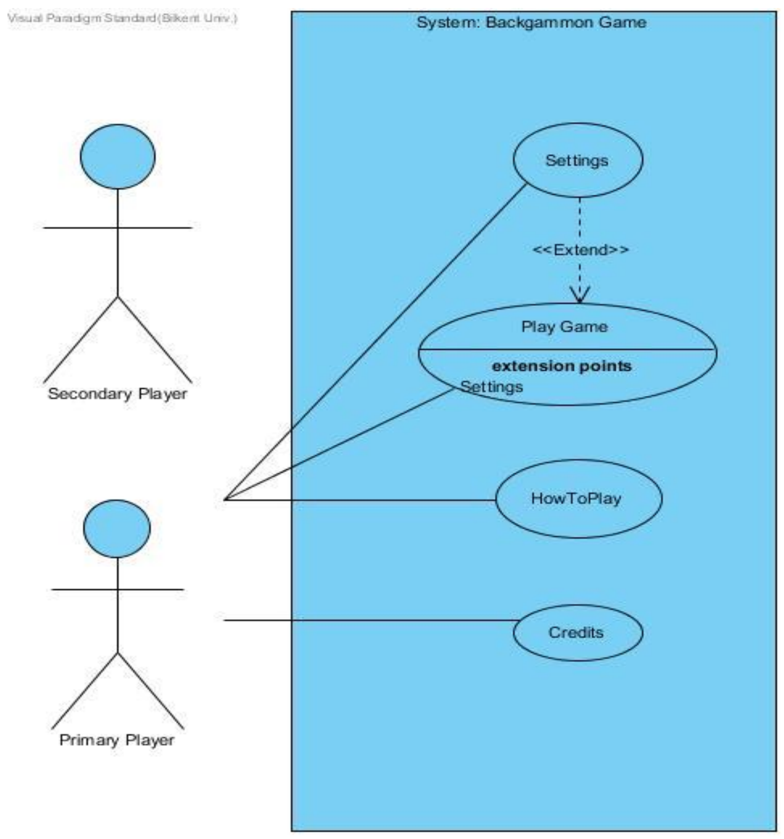


Figure 4.1 – Illustrates the Use Case Model

4.1.1. Play Game

Use Case Name: Play Game

Primary Actor: Player

Stakeholders and Interests:
-Player aims to defeat his opponent
-System keeps the score of each players .

Pre-condition: For first running, game settings are set as default. If Player changes game settings, adjusted settings will be saved and used by System.
Post-condition: -.

Entry Condition: Player selects “Play Game” button from Main Menu.
Exit Condition: Player selects “Return to Main Menu” from Pause Menu.

Success Scenario Event Flow:
1.Game is started by System.
2.Players decide how many round they will be playing and change their names.
3.Players plays the game until one of them wins the round.
4.System increments the score of the corresponding player.
5.Next round begins.
Players repeats the steps 3 – 5 until one of the player wins by best of rounds.
6.System returns to Main Menu.
Player repeats the steps 1 – 6 if he wants to play game again.

Alternative Flows:
3A. Players start to play the round:

3A.1. Each player roll a die.
3A.2. System decides who wins and the winner starts playing.
3A.3. Starting player rolls 2 dice.

3A.4. Player clicks a checker of his desire by mouse.

3A.5. System shows the player's checker's playable moves.

3A.6. Player decides where to play his checkers.

3A.7. Player's turn ends and passes turn to the other player.

- *Players follows steps 3A.3 – 3A.7 to make move.*

3B. Player tries to make a hit:

3B.1. Player's checker and his opponent's checker collides on the same space.

3B.2. System checks the move's availability. If the opponent's checker is alone in the space, Player's checker takes its place and opponent's checker will be taken to the bar. Otherwise, system warns player that move is unavailable.

3B.2. Player ends the turn and turn passes to other player

3B.3. Player rolls 2 dice.

3B.4. System checks the availability of move of the hitted checker. If it is available, system marks the playable spaces, else player passes turn without making any moves.

- *Whenever a player tries to make a hit, steps 3B1-3B4 are applied.*

3C. Player is in bearing-off phase:

3C.1. Player's all checkers is located in the final six spaces of the board.

3C.2. Player roll 2 dice.

3C.3. System checks the available checkers to collect and mark them.

3C.4. Player collects checkers located in the corresponding spaces. If there is no available checkers, player have to move forward as in the *3A.3- 3A.7* or if there is no checker in the bigger spaces than the dice values, player can collect checkers from the more significant space.

- *Players repeat the steps 3C.1- 3C.4 until one of the player collects all of his checkers.*

3C.5. Player who collected first wins the round.

3C.6. System increment the score of the corresponding player. If winner wins the round before his opponent cannot reach the bearing-off phase, System increment the winner player’s score by 2.

4.1.2. View Credits

Use Case Name: View Credits

Primary Actor: Player

Stakeholders and Interests:

- Player wants to learn about names and contact information of creators of BackGammon.

- System displays names and contact information of game’s developers.

Pre-conditions: Player should be in Main Menu.

Post-condition: -

Entry Condition: Player selects “View Credits” button from Main Menu.

Exit Condition: Player selects “Back” button to return previous menu.

Success Scenario Event Flow:

1. System displays names and contact information of developers of Backgammon.

Alternative Flows:

A. If player wants to return main menu:
   a. Player selects “Back” button to return Main Menu.
   b. System then displays Main Menu content

4.1.3. Change Settings

Use Case Name: Change Settings

Primary Actor: Player

Stakeholders and Interests:

- Player wants to change game's settings: sounds on/off, changing checker's colour.

-System updates the configuretions according to change by player.

Pre-condition: At the first run, game settings will be set to the defaults. If player make some changes, these preferences made through settings windows will be used by the system.

Post-condition: Game settings are updated according to change.

Entry Condition: Player selects "Change Settings" button from menu.

Exit Condition: Player selects "Back" button to return menu.

Success Scenario Event Flow:

1. Player presses "Change Settings" button to make some adjustments about game's settings.

2. Game settings are displayed to player in "Change Settings" screen.

3. Player adjust settings according to his/her preferences.

4. System updates game's settings according to player's preferences.

Alternative Flows:

A. If Player desires to switch back to default settings at any time:

A.1. Player selects "Defaults Settings" button on "Change Settings" screen.

A.2. System sets settings back to default settings.

B. If Player wants to return back to the previous screen at any time:

B.1. Player selects “Back” button from “Change Settings” screen.

B.2. Game settings are updated.

B.3. Player returns to the previous menu.

4.1.4. View Help

Use Case Name: View Help

Primary Actor: Player

Stakeholders and Interests:

- Player who wants to learn the basics of BackGammon and how to play it.

-System shows a text document explaining the basics of BackGammon and how to play it.

Pre-condition: Player should be in Main Menu or Pause Menu

Post-condition: -

Entry Condition: Player selects “View Help” from Main Menu or Pause Menu

Exit Condition: Player selects “Back” to return previous menu.

Success Scenario:

1. Player selects “View Help” from Menu.

2. System displays help document that informing player with basics of the game and Backgammon’s rules.

Alternative Flow:

A. If Player wants to return previous menu:

A.1. Player selects “Back” button from “View Help” screen.

A.2. Player returns previous menu.

4.2. *Dynamic Models*

This section provides detailed information about The Backgammon Game. Section consist of sequence diagrams, activity diagram and object class diagrams. The game has some cruicial objects. Such as checkers and board. In the sequence diagram, our goal is to express how the project flows.

4.2.1. Sequence Diagrams

4.2.1.1. Set Game

Following sequence Diagram illustrates the scenario explained below:

<u>Scenario:</u> Players Abdullah and Ali (one of them decided as a main player) request to start the game by pressing the " Play Game" button in the Main Menu. After that System initializes a new screen which players need to change their names and choose how many rounds will they play. Then System sets the game objects with the corresponding data and players presses start game button in the Set Game Menu.

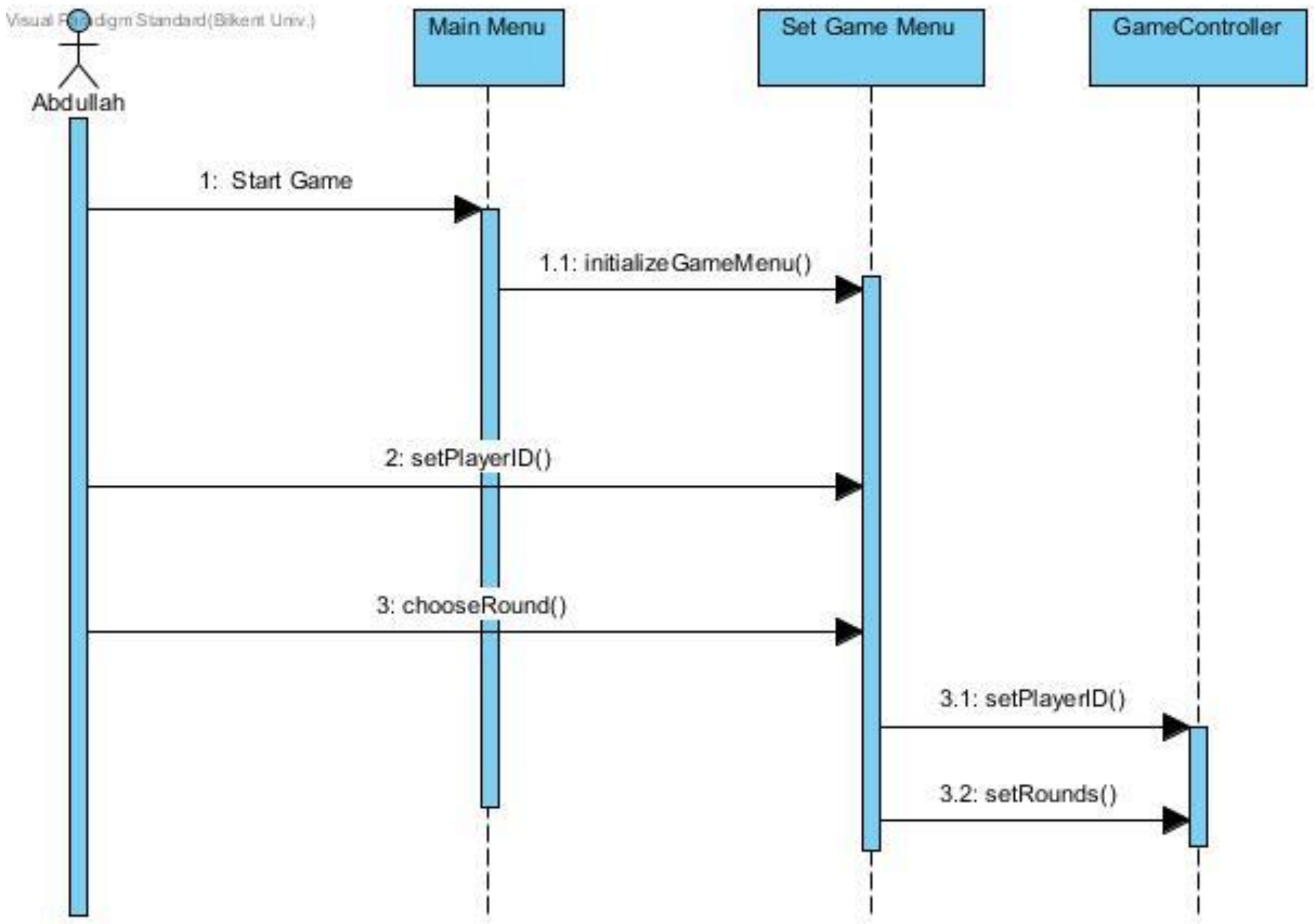
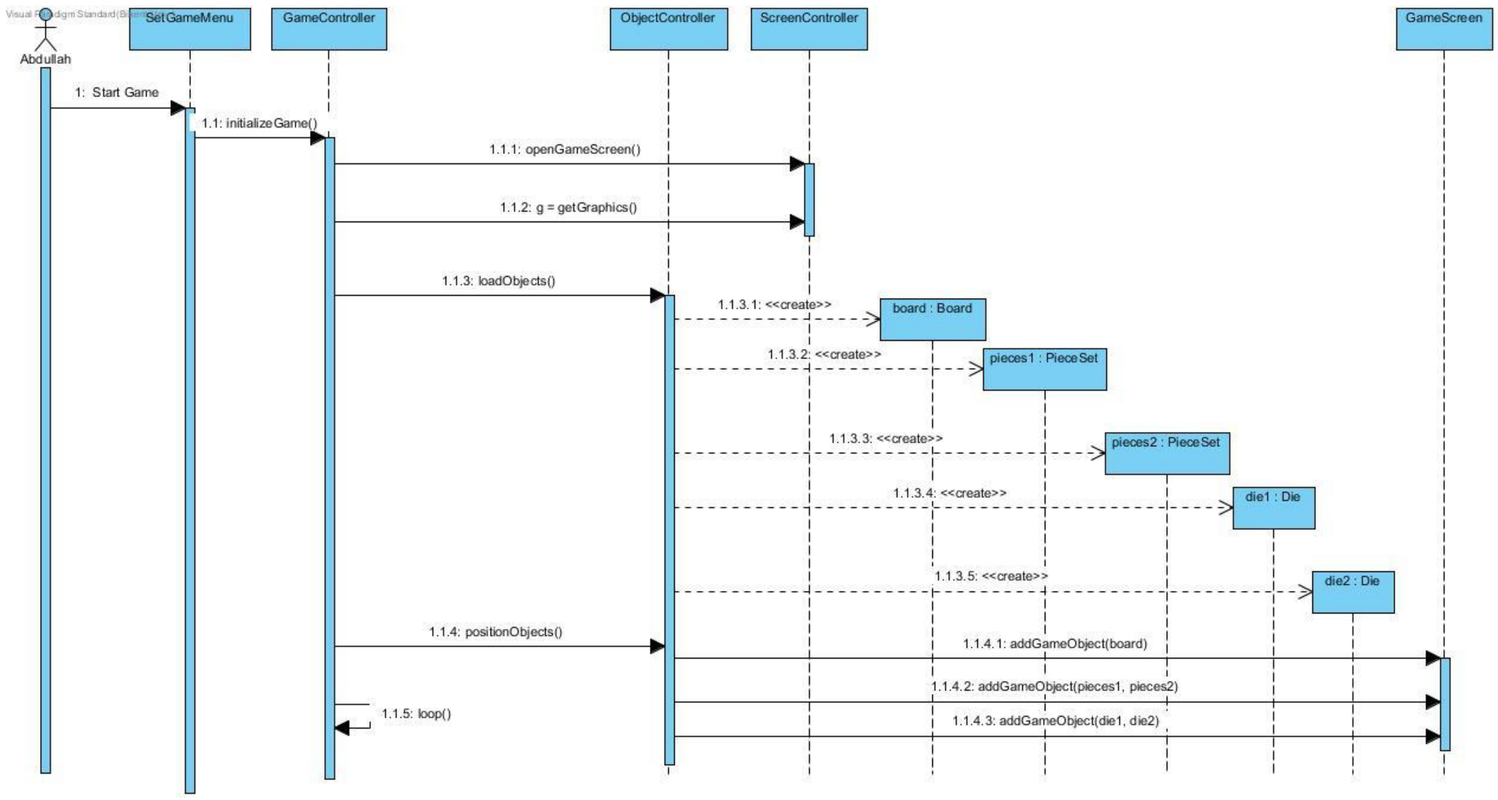


Figure 4.2.1.1 shows the first case of the game

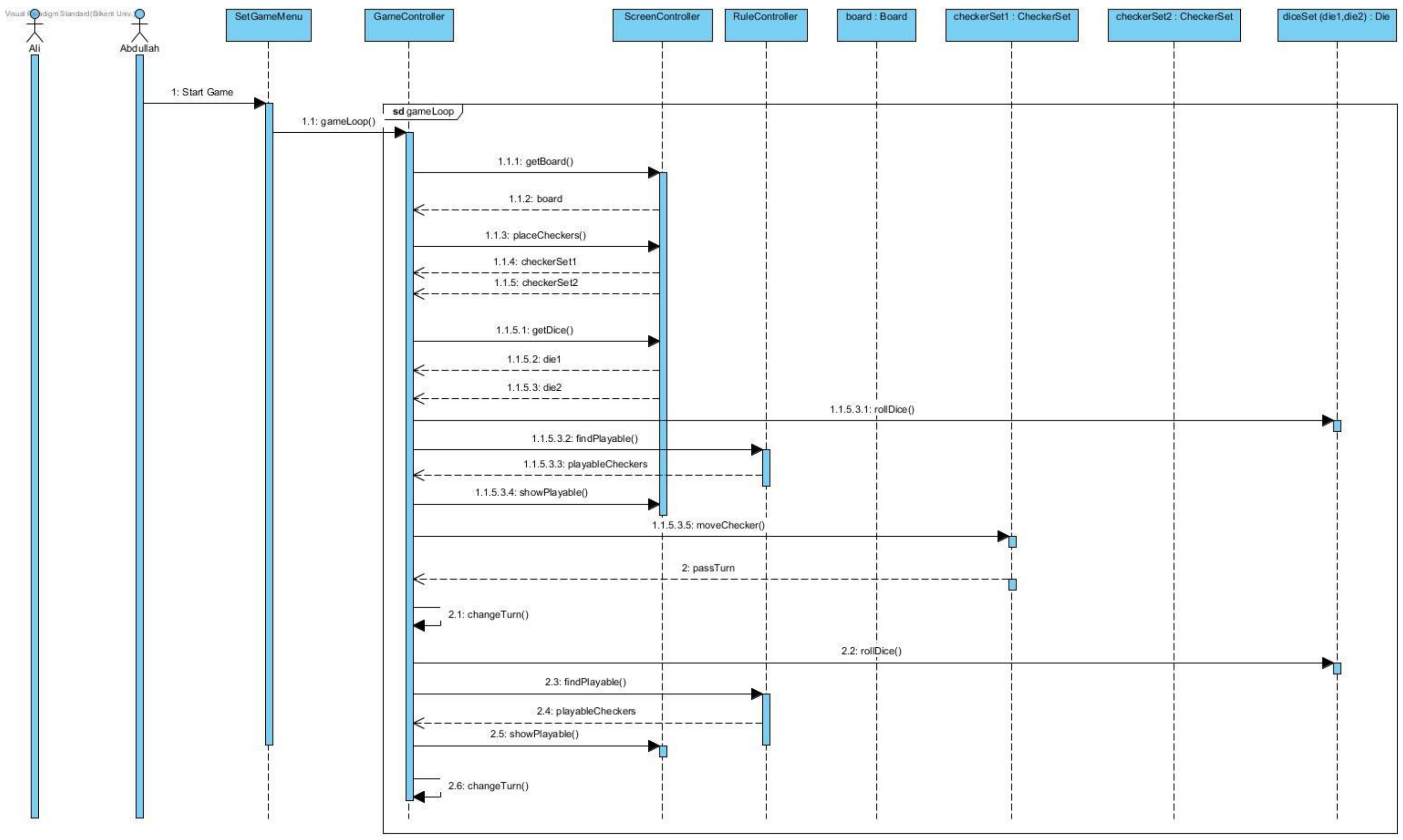
4.2.1.2. Initialize Game

<u>**Scenario**</u>: Player Abdullah wants to initalize the game pressing 'Play Game' button in Main Menu. The GameController opens the proper game screen and creates needed graphics. Then, GameController calls 'loadObjects()' to create game board, checkers and dice. Then it positions all the objects into their initial states by connecting them to game board and defines other relations. Finally, it sends all of these objects to the GameScreen.



4.2.1.3. Initialize Game

<u>**Scenario**</u>: Players Abdullah and Ali plays the game. For each move game makes a loop until one player’s all checkers will be collected. Players, after game initialize, starts to playing game. In the game loop, System firstly gets the board data and place checkers of both players. Then players sequentially, roll dice and click a checker to decide where to move the corresponding checker. After clicking a checker, System first calculates playable slots and shows it the to current player. Player moves its checker and System passes the turn to the other player and game loops until no checkers left.



4.2.1.4. Change Settings

Following sequence diagram illustrates Scenario 1 given below:

<u>**Scenario 1:**</u> Armağan wants to make game setting changes. He presses change settings button from either MainMenu or PauseMenu windows. Then, the System displays settings and Armağan changes settings according to his preferances in ChangeSettings Menu. After pressing back button, System's settings are updated according to preferences of Armağan by GameManager.

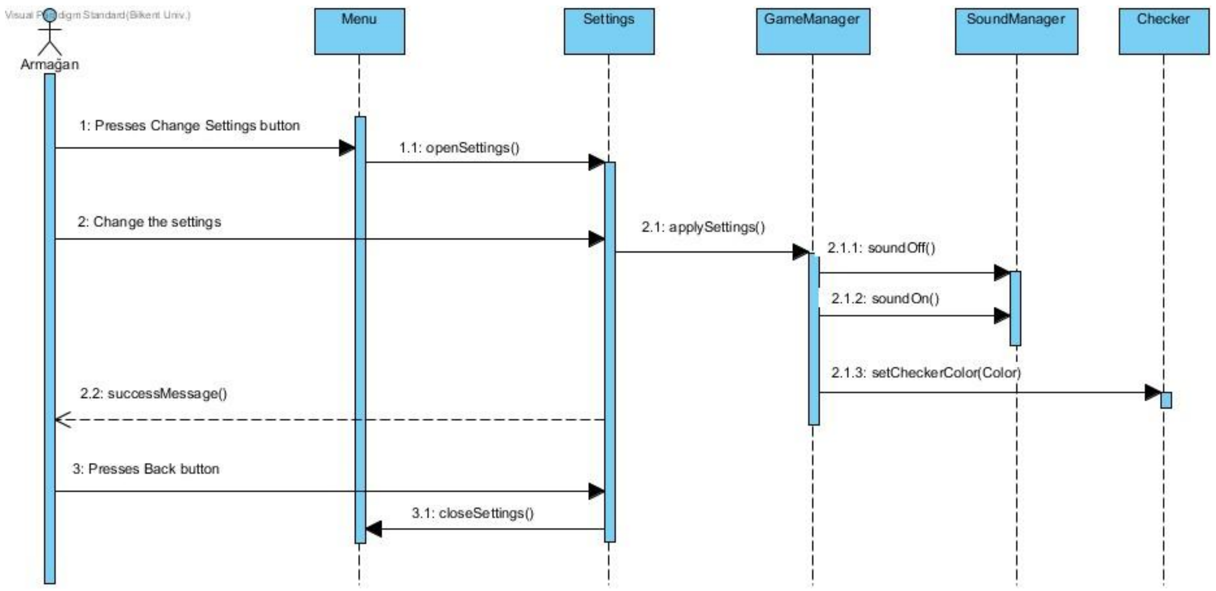
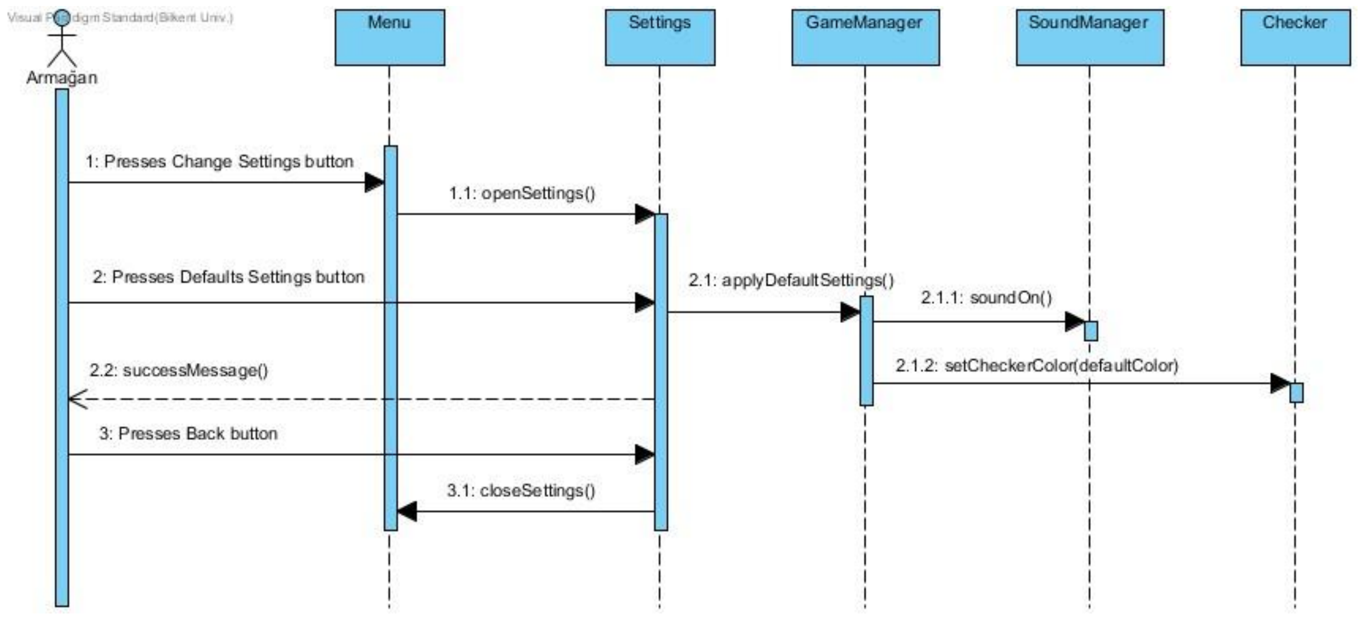


Figure-4.2.1.4 illustrates Change Settings sequence diagram

GameManager sets setCheckerColor(Color) according to player’s preference and sends this to Checker class this parameter. New settings according to Sound are sent to SoundManager to be applied by GameManager.

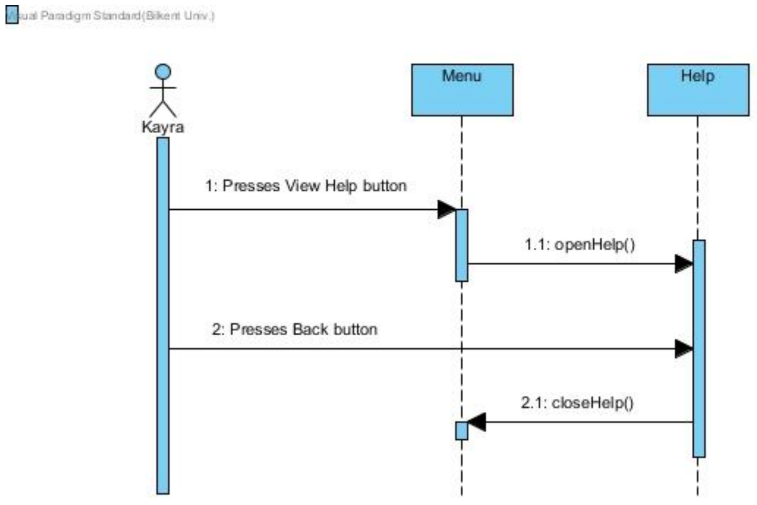
Following sequence diagram illustrates Scenario 2 given below:

Scenario 2: If Armağan wants to use default settings, open Change Settings from MainMenu. Then in Change Settings Menu he can press default settings button to return back into predetermined settings of the game by the System. System displays the new window of settings according to default settings.


Visual Paradigm Standard(Bilkent Univ.)
Armağan
Menu
Settings
GameManager
SoundManager
Checker
1: Presses Change Settings button
1.1: openSettings()
2: Presses Defaults Settings button
2.1: applyDefaultSettings()
2.1.1: soundOn()
2.1.2: setCheckerColor(defaultColor)
2.2: successMessage()
3: Presses Back button
3.1: closeSettings()

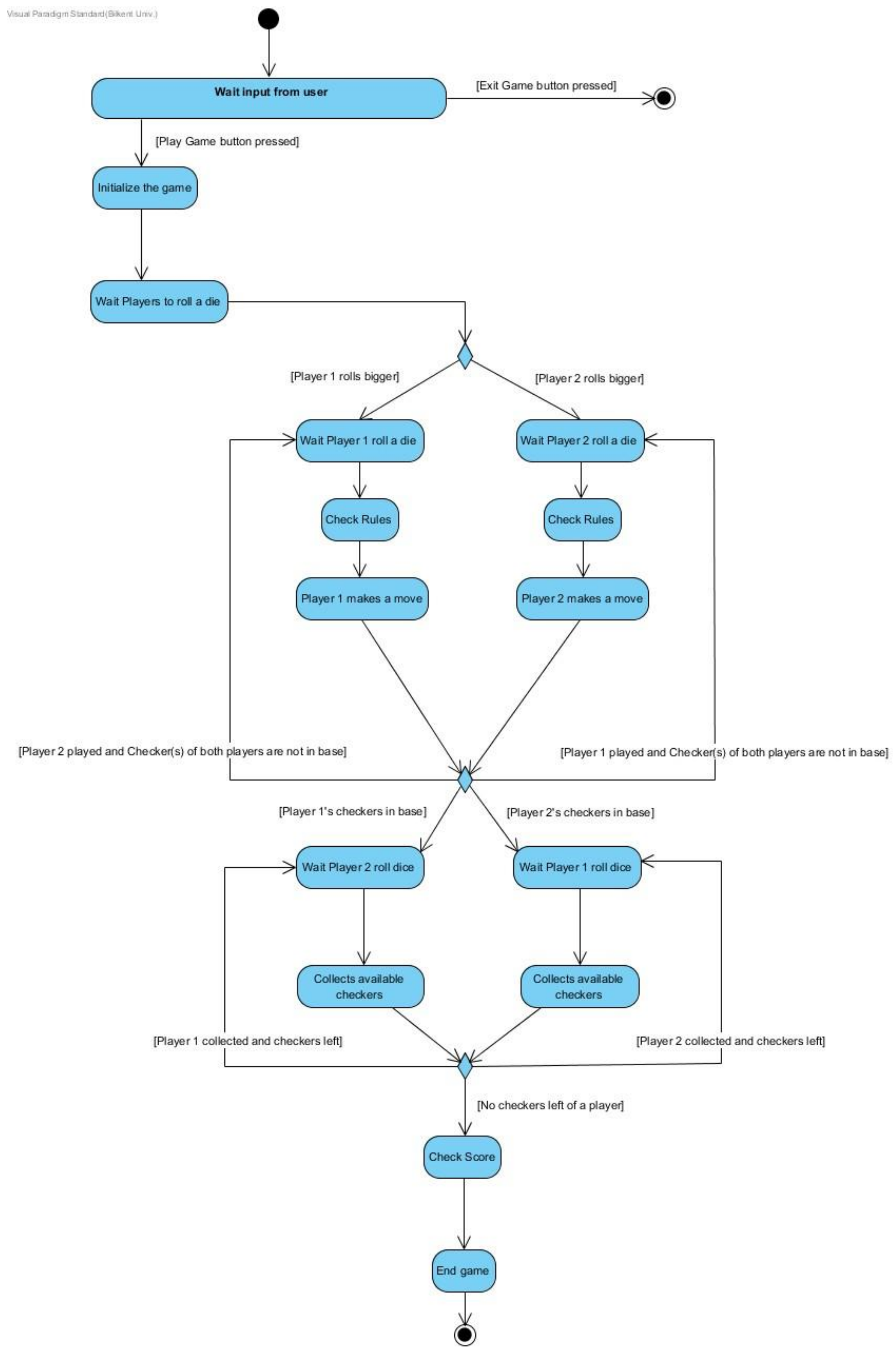
4.2.1.5. View Help

Scenario: Player Kayra decides to look at the how to play section. To achieve this, she clicks the “ View Help” button in the main menu. Then System, initializes the “ Help” on the screen. Finally, Kayra decides to leave this section and enters the main menu again.


Visual Paradigm Standard(Bilkent Univ.)
Kayra
Menu
Help
1: Presses View Help button
1.1: openHelp()
2: Presses Back button
2.1: closeHelp()

4.2.2. Activity Diagram

This is the diagram of the main system of the game;

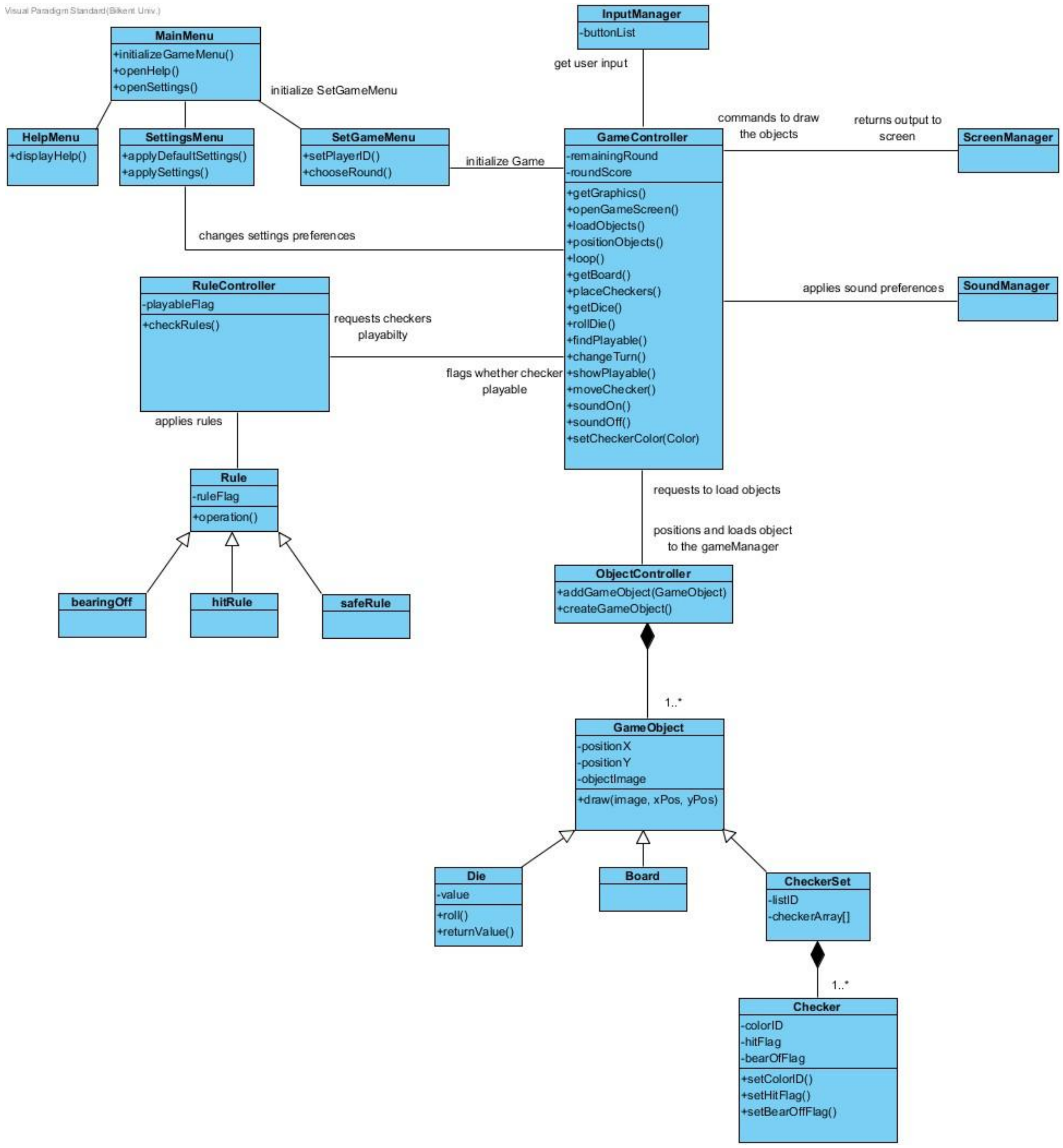


Visual Paradigm Standard(Bilkent Univ.)
Wait input from user
[Exit Game button pressed]
[Play Game button pressed]
Initialize the game
Wait Players to roll a die
[Player 1 rolls bigger]
[Player 2 rolls bigger]
Wait Player 1 roll a die
Wait Player 2 roll a die
Check Rules
Check Rules
Player 1 makes a move
Player 2 makes a move
[Player 2 played and Checker(s) of both players are not in base]
[Player 1 played and Checker(s) of both players are not in base]
[Player 1's checkers in base]
[Player 2's checkers in base]
Wait Player 2 roll dice
Wait Player 1 roll dice
Collects available checkers
Collects available checkers
[Player 1 collected and checkers left]
[Player 2 collected and checkers left]
[No checkers left of a player]
Check Score
End game

When one of the players press Play Game button the system initializes the game objects such as Checkers, Dice, Board. When this initialization is over, system waits for the players to roll a die to determine which player is going to start the game. One of the player starts with rolling dice to know how many slots he/she can move . Player sees the available slots according to the Rules of the game and choose one of the possible moves. The system checks if any of the players have their checkers in their base and checks which player moved last time to determine the playing order.

If one of the players have his/her checkers in base, then he/she can start to bear off some checkers at next playing order according to the new values come up from dice. Every player has to collect checkers in base in order to bear off their own checkers. The one that finishes bearing off checkers win the round and if there are no more rounds, win the game. The system continues in a loop to maintain the game continuity and ends when the rounds finish.

4.3. *Object and Class Model*



The object and class model of the Backgammon Game. The game consist of 19 objects.

The Menu classes, generally consisting of reaching sections and navigate the user on the program.

Core of the model is the controller classes. These classes actually controls the game loop, settings and the GUI of the game. The classes explained briefly below.

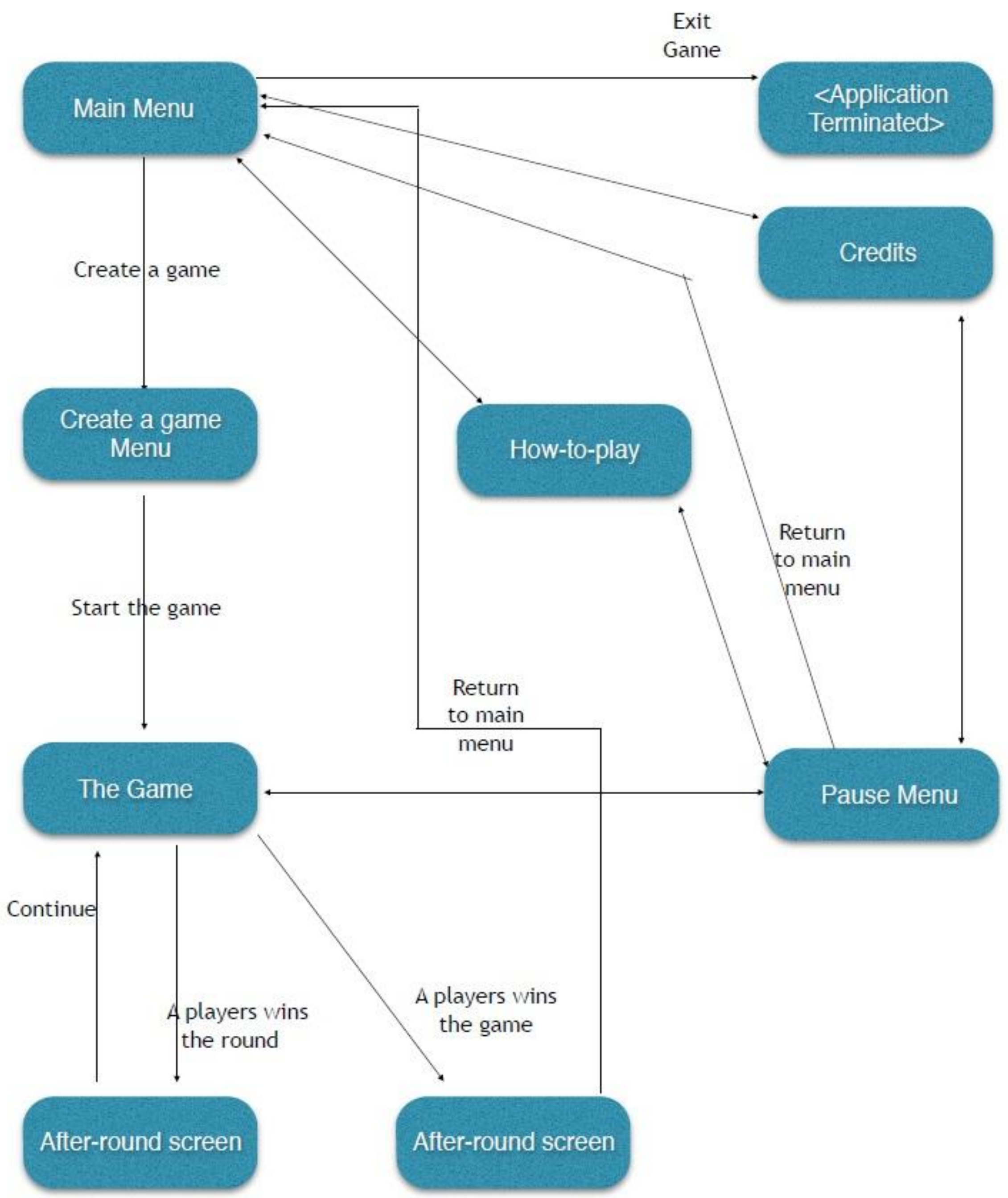
- GameController: This class controls the gameloop and connect the user imput with the other controller classes.
- ObjectController: With this class, GameController demands to create and load the game objects to the System.
- ScreenManager: This class creates the graphical interface on the screen by managing Java methods.
- SoundManager: This class controls the sounds in the game.
- RuleController: In the game loop, System checks the available checkers and the moves validity. Basicly, this class controls whether a move is valid or not.

The structure of the game Objects inherit GameObject. The objects controlled by ObjectController. Classes explained briefly below.

- Rule: This class is the parent class of the rules and contains as rule classes, such as bearingOff, hitRule, safeRule.
- GameObject: The parent class of the actors in the game. Such as, Die, Board, CheckerSet and Checker

5. User Interface

5.1. Navigational Path



5.2. Screen Mock-ups

5.2.1. Screen Mock-ups

Main menu is the first thing to be seen right after opening the application. In our application, it is the most extensive menu. It allows user to navigate to create a new game menu, how-to-play menu, credits, adjust the sound and essentially, exit the game.

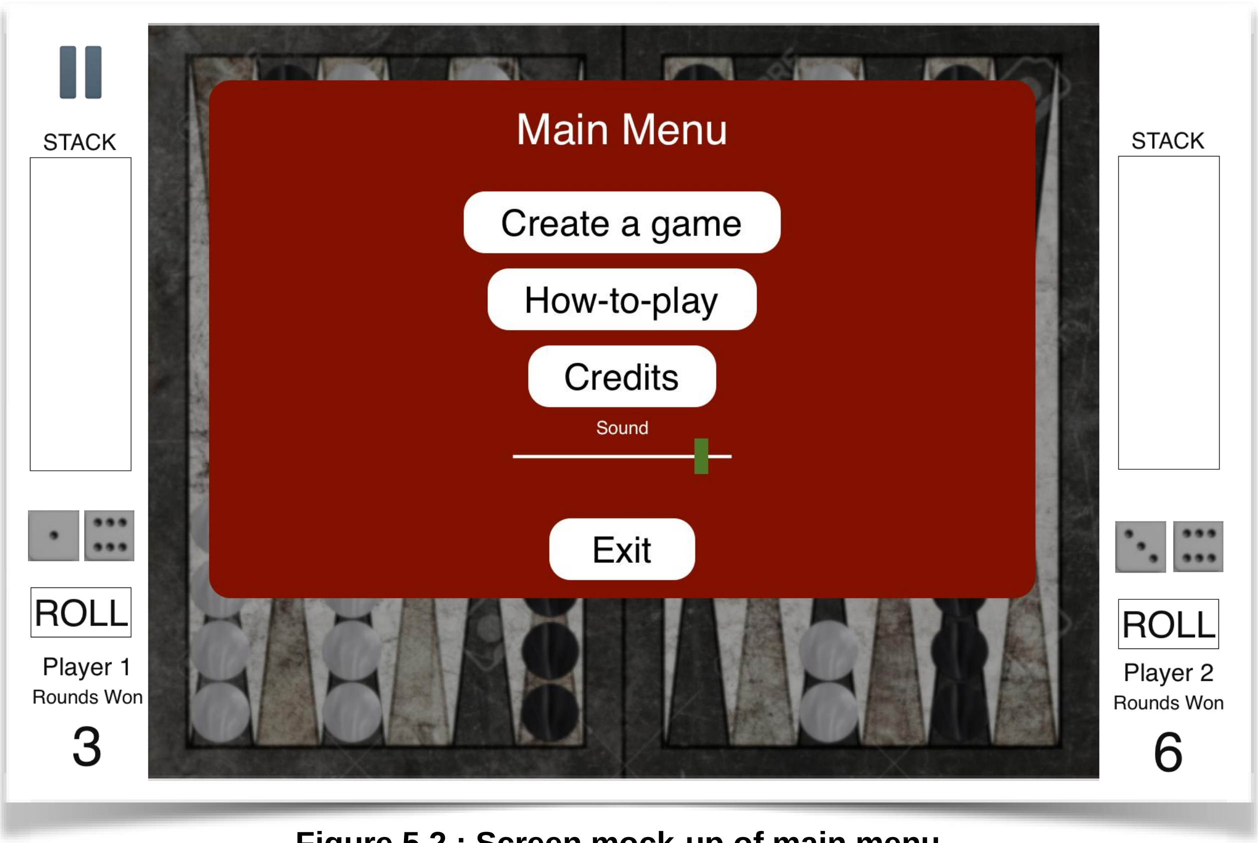


Figure 5.2 : Screen mock-up of main menu

Create Game: In this case, Player is about to start a two-player game. Nicknames of players, number of rounds are asked and two dices are about to be rolled. See figure 5.3 for more.

How-to-play: Includes instructions about backgammon.

Credits: Leads to a credits page.

Exit: Terminates the application.

5.2.2. Create a Game

Default values for nicknames are Player 1 and Player 2, respectively. The question mark inside the white square represents an unrolled dice. After ROLL buttons are pressed, the die will show up and will determine which player goes first. Color of chips are selected here but also could be changed later on.

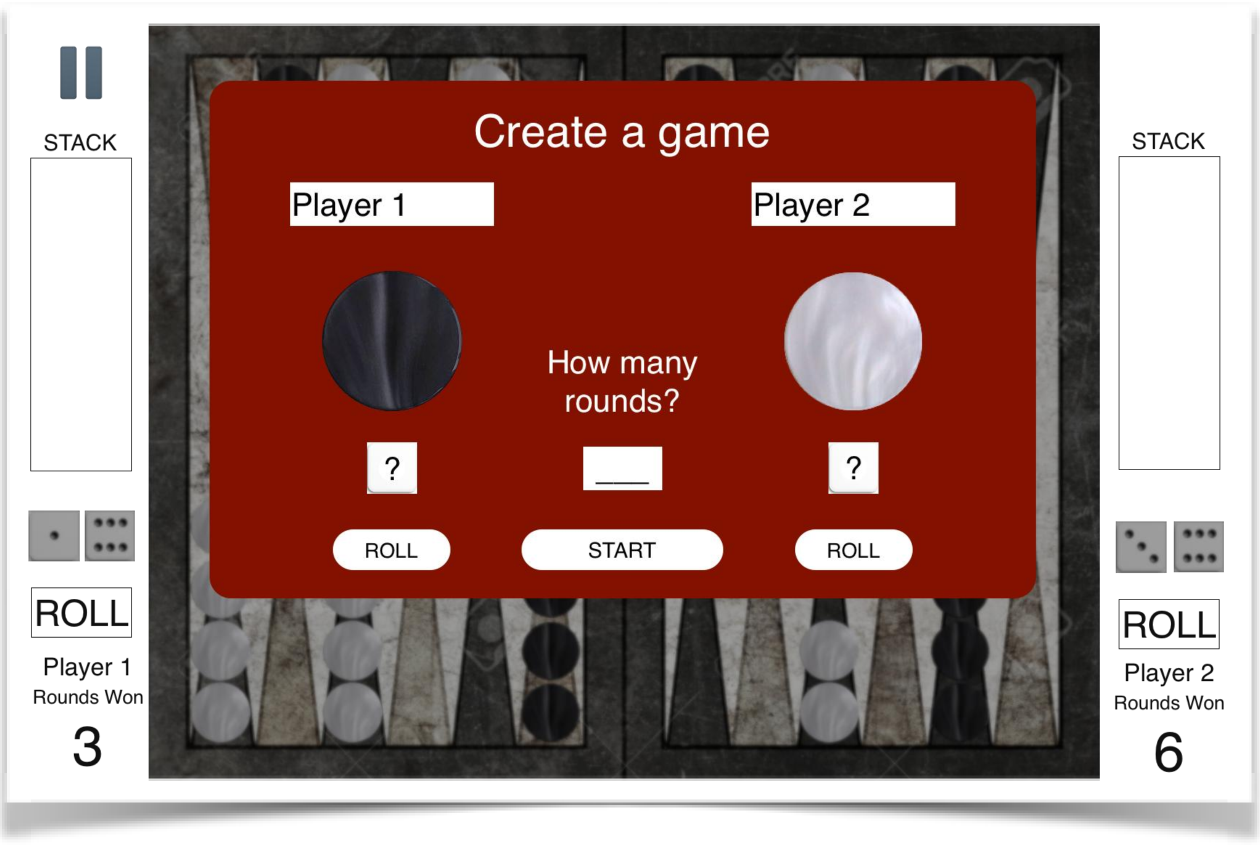


Figure 5.3 : Screen mock-up of Create a Game menu

5.2.3. The Game

STACKs will hold chips that have been picked up from the board. Once one of them is full (i.e. has 15 chips), the round would be complete. Two dices right below the stacks shows last 2 dices each player has rolled. Each player ROLLs his/her dice by pressing the button on their side. Once the rounds won reached a specified number, the whole game would be complete.

Once each round is complete, the game goes into “end of the round screen”.

If the game is complete, the game switches into “after game screen”.

Blue button on the top left corner leads to the pause menu.

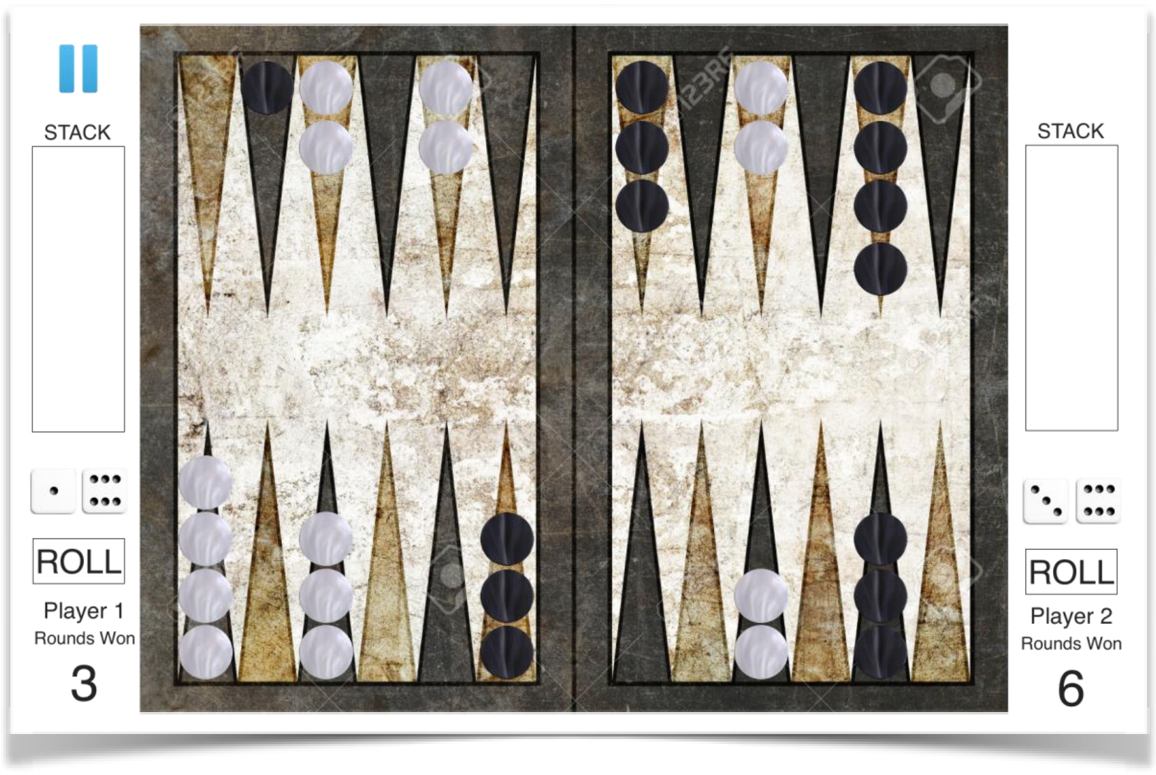


Figure 5.4 : Screen mock-up of Gameplay

5.2.4. After-round Screen

After each round, until the specified number of rounds reached, game switches into this screen.

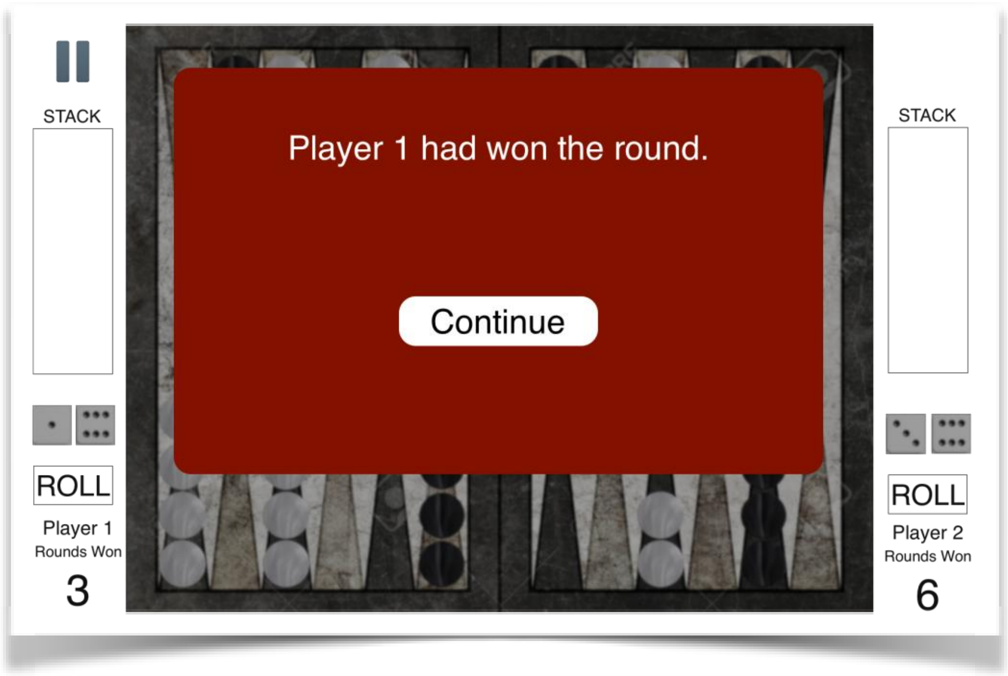


Figure 5.5 : Screen mock-up of an After-round screen

5.2.5. After-game Screen

Once the specified number of rounds had reached, game switches into this screen and that's when a game is finished.

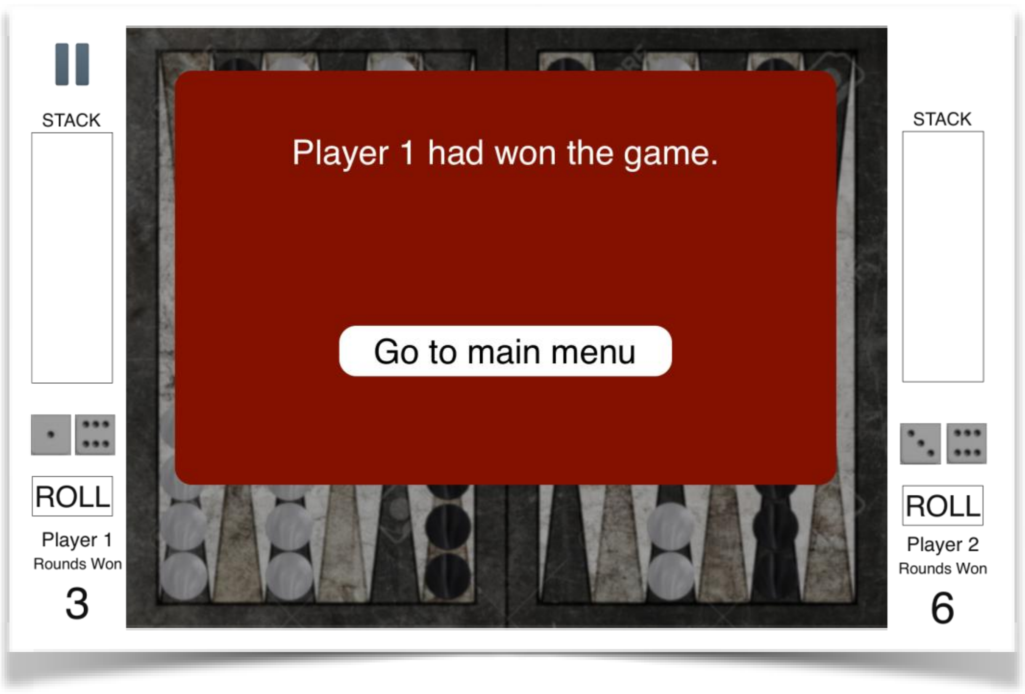


Figure 5.6 : Screen mock-up of an After-game screen

5.2.6. Pause Menu

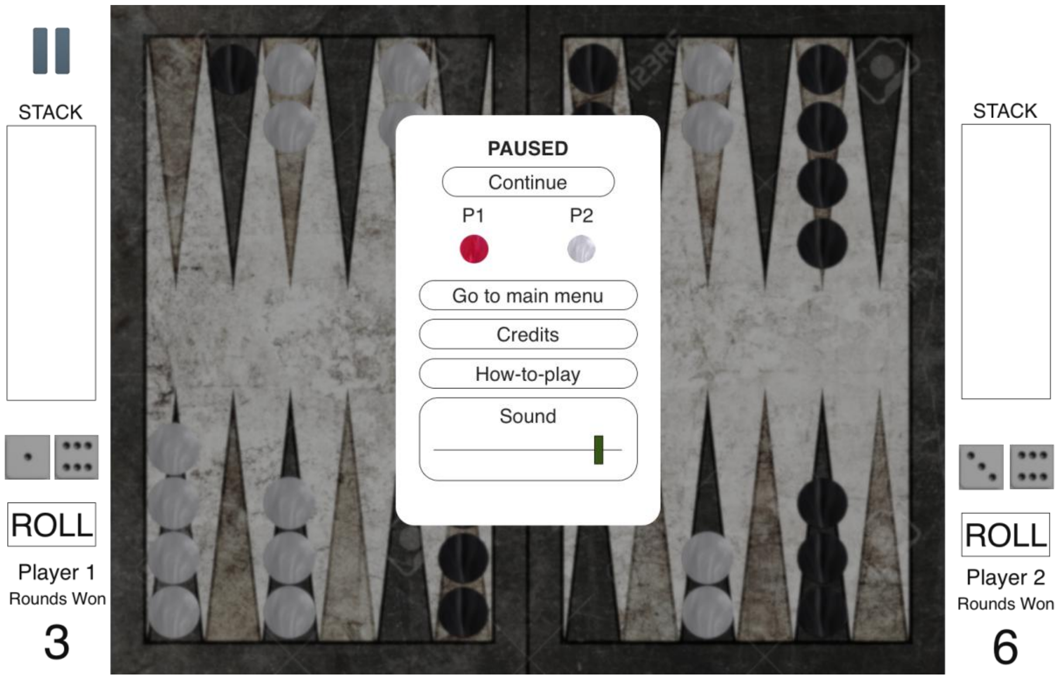


Figure 5.7 : Screen mock-up of a pause menu

Pause menu is the only way to intervene with the game except the game elements here. As chosen in the create a game menu, player can also change the checker colours here. There are links to switch into Credits and How-to-play, as it did in the main menu. Also can adjust the sound here, and also navigate to main menu.

5.3 Other Ui Elements

5.3.1 Checkers

Three colors: Black, White and Crimson

5.3.2. Die

Six forms of die are as follows.

7. Conclusion

We have projected our understanding of a two player backgammon game into this analysis report. We did that in terms of UML models, diagrams and textual descriptions.

In our use case model, we specified how a user can move through the software. Based on that, we modeled sequence diagrams on Visual Paradigm. To show the workflow, we constructed activity diagrams, they serve a functionality of visualizing the computational workflow. After we got all those important UML models as reference, we modeled the class diagram. Then, we decided on navigations between screens and built a navigational path out of it. Considering the navigational path, we created the UI with buttons for navigations.

This analysis report will serve as a blueprint for our upcoming programming days. That is because we have decided on important design decisions and implementation plans. We are going to experience how such a document can help us during coding parts.