



CS 319 - Object-Oriented Software Engineering

Fall 2020

Project Analysis Report Iteration 2

Monopoly

Group 3D

Cansu Moran - 21803665

Elif Gamze Güliter - 21802870

Melisa Taşpınar - 21803668

Öykü Irmak Hatipoğlu - 21802791

Yiğit Gürses - 21702746

Table of Contents

Table of Contents	2
1 Introduction	5
2 Overview	5
2.1 General Features	5
2.2 Editing Feature	6
2.3 Board	6
2.4 Properties	6
2.5 Joker Places	7
2.6 Chance and Community Chest Cards	8
2.7 Playing the Game	8
2.7.1 Starting	8
2.7.2 Game Actions	8
3 Functional Requirements	9
3.1 Requirements about Game Play	9
3.1.1 Landing on Property Squares	9
3.1.2. Landing on Special Squares	10
3.1.3 End of the game	11
3.2 Requirements about Editing the Game Features	11
3.2.1 Editing Properties	12
3.2.2 Editing Joker Squares	12
3.2.3 Editing pictures	12
3.2.4 Editing Miscellaneous Features	13
3.3 Additional Functional Requirements	13
4 Non-Functional Requirements	13
4.1 Usability	14
4.2 User Friendliness	14
4.3 Portability	15
4.4 Additional Requirements	15
5 Pseudo Requirement	15
6 System Models	16
6.1 Use Case Model	16
6.1.1 Use Case Name: Play a Game	16
6.1.2 Use Case Name: Interact With Property	18
6.1.3 Use Case Name: Edit a Board	18
6.1.4 Use Case Name: Edit A Square	19
6.1.5 Use Case Name: View Help	20

6.1.6 Use Case Name: Change Settings	21
6.1.7 Use Case Name: Display Credits	22
6.2 Dynamic Models	22
6.2.1 Sequence Diagrams	22
6.2.1.1 Editing the Game : Properties	22
6.2.1.2 Landing on a Property and Buying it	23
6.2.1.3 Landing on a Joker Square and Paying Rent	24
6.2.1.4 Landing on Community Chest and Chance Squares	25
6.2.2 Activity Diagram	28
6.2.2.1 Play a game	30
6.2.2.2 Edit a Board	32
6.2.3 State Diagrams	33
6.2.3.1 Player State Diagram	33
6.2.3.1.1 Property State Diagram	35
6.2.3.1.2 Color Group State Diagram	37
6.3 Object and Class Model	38
6.4 User Interface	42
6.4.1 Navigation Path	42
6.4.2 Screen Mockups	43
6.4.2.1 Main Menu	43
6.4.2.2 Player Selection Screen	44
6.4.2.3 Board Selection Menu	45
6.4.2.4 Settings Screen	46
6.4.2.5 Credits Screen	47
6.4.2.6 Help Screen	48
6.4.2.7 Edit Screen	49
6.4.2.8 Play Screen	52
6.4.2.9 Property Square Dialogs	54
6.4.2.10 Auction Dialog	55
6.4.2.11 Sell Dialog	56
6.4.2.12 Joker Square Dialog	57
6.4.2.13 Chance Square Dialog	58
6.4.2.14 In-game Menu	59
6.5 Priorities of Requirements	60
7.Improvement summary	60
8 References	61

1 Introduction

Our group 3D, decided to implement the board game “Monopoly” with the name “Monopoly Worlds” for our CS 319 project. The game’s objective can be predicted from its name, where players aim to become the one and only surviving player in the game by making other players bankrupt by buying all the properties, like the economic term monopoly, where there is a single presence dominating the market. Due to its numerous aspects such as properties, tokens (the pawns in Monopoly), chance and community chest cards, and money, it is a suitable game for implementing Object Oriented programming. Since it is a game we played during our childhoods and we know all the rules by heart, we chose to design Monopoly. However, we decided to add a function that will elevate the game in many ways such as creativity and flexibility.

What makes Monopoly Worlds different from the original one is the fact that we are introducing an edit feature to Monopoly. There are various different themed Monopoly versions on the market such as Monopoly Turkey version, or TV characters version and if you want to play them, you should purchase all the themes you want separately. However, with our edit feature addition to Monopoly, the game can be edited in any way that the players wish. You can design your own version of Monopoly like Monopoly Space or Monopoly Bilkent. When you build your own version of Monopoly, you can save it and play it with your friends.

2 Overview

2.1 General Features

In our project, we will be implementing a computer version of the board game Monopoly. The game will not be played online, instead, it will be implemented as a one-computer game where players will play in turns. Different from the original Monopoly, our version can only be played by 2-4 players. We decided not to use the original limit, which is 2-8, because as the number of players increase, players will start to lose their attention since they will start to wait more for their turn to come. Therefore, we decided to limit the number of players to at most 4.

At the beginning of the game, a version of Monopoly that is similar to the board game will be offered to players. The players can either play this default version or they can go to the editor to create their own versions using the default one as a template. After saving their versions, users can still edit them and create new versions using their saved games as templates, or they might

update their already existing versions. Users can play on any board that they have saved. The default version of the game has similar rules and squares to the original Monopoly. However, with the editor, the users can edit the neighborhoods and they can add different squares with properties that are not necessarily included in the original Monopoly.

2.2 Editing Feature

The most distinguishing feature of our version of Monopoly is the editing feature which enables players to play the Monopoly theme they want to play by allowing them to design their games. The edited boards can be saved to be played with later. From the currency on the game to the picture on the chance square our version offers many features to edit and customize the game. Users can change properties and joker squares and also add as many different types of squares as possible which enables different approaches to Monopoly. The users can also change the appearance of the board, squares and pawns if they are planning to use a theme on the design.

2.3 Board

The Monopoly board consists of 40 squares, and all the squares except from the start square can be one of four different square types: Property, Joker, Chance and Community Chest. With the help of the editing feature of our version, the user can design the board whichever way he/she likes.

In the edit section, the board designed by the user can be saved to be used later or players can immediately play on the board that was just designed. Additionally, players can choose the board they want to play on from the boards that have been saved earlier or the default boards provided on the game. When editing the board, the user can be guided by the design of the default board.

2.4 Properties

When editing the board, the user can also edit the properties. The user can change the names, color groups, and the locations of the properties. The user can also change the money rates of the properties, for example, the mortgage rate, which is for what percent of the buying price

should the property be mortgaged to. Other rates that can be changed are the hotel and house price and rent amount.

2.5 Joker Places

There are certain squares in the original Monopoly that we decided not to include in our version. Instead, we decided to create a special Joker square that can be edited into a similar square to the ones included in original Monopoly or can be edited to be a completely different square.

In traditional monopoly, there is “Jail” where players must go wait for a couple of turns if they land on the “Go to Jail” block. Furthermore, there are two other special squares called Income Tax where players pay an amount of tax if they land on it and Free Parking where no action such as paying tax or paying rent is required. The Joker square in some sense is a combination of these squares in the original Monopoly. Instead of a classic jail, income tax or free parking places, there will be Joker squares that can have 3 different properties. These properties include winning/losing money, having to wait for a certain number of turns and moving a certain number of squares ahead. Using the editing feature , the user can create different Joker squares using a combination of these features.

These Joker places can be made into a Jail, if the user adds a certain number of turns to be waited for players who land on this square. They can also be made into an Income Tax square by adding money actions to that square. They can also be made into a Free Parking square by not adding any money or movement actions. The features of these Joker squares will be up to players, therefore, if they prefer, they can have no jail in the game or they can have multiple jails or even have a jail where if you go to jail you earn/lose money. Unlike the property squares, Joker places cannot be bought.

2.6 Chance and Community Chest Cards

There are two kinds of cards in the game, chance and community chest cards. These cards will be the same format as the original monopoly game, and they will not be editable. Only the

currency that appears on the cards will be automatically updated according to the edited currency name in the editor.

2.7 Playing the Game

2.7.1 Starting

At the beginning of the game, the user is presented with Edit and Play options. Play option takes the user to a screen where they can select which board they would like to play on. After board selection, the number of players should be selected. Following that, players will enter their usernames in order. The first player who wrote her/his username will start the game and other player's turns will come in the same order with their names. Every user will be assigned a color of their choosing so that after they buy a property, the color of that property's square on the board will change to the player's color. The players will also be able to choose one of four pawn options that comes with the board they have chosen.

Every user will receive money at the beginning of the game i.e. 1500 \$ which can be edited in the Editor if players ever want to start with different amounts of money (see section 3.3). All the players' tokens start from the Start square and every time they pass the Start square, they will receive a designated amount of money. The game begins when the first player rolls the dice.

2.7.2 Game Actions

The game will be auto controlled, which means that users will not have to move their tokens, they will only roll their dice. According to the number that comes out from dice, users' tokens will be moved to the designated square automatically.

The squares on the board can either be properties or special squares such as chance, community chest and Joker. According to the features of the square that the tokens arrive, players will be able to perform certain activities.

3 Functional Requirements

3.1 Requirements about Game Play

3.1.1 Landing on Property Squares

Buying a property:

- Players will be able to buy a property that they land on using the buy button associated with the square. However, they can also buy a property from auction which will be explained in additional requirements.
- After buying that property, the color of the square will change into that player's color. In the board game version, players need to keep cards of the places that they own. However, in this version, they will not hold cards, instead, the places that they own will be indicated by their colors on the board.

Paying rent:

- If a player lands on an unmortgaged place owned by other players, rent is taken from the user and is transacted to the owner automatically by the game.
- Rents increase with a predetermined percentage if the owner has completed all the properties of that color group and has houses and hotels built on that property and players will be able to change this percentage with the editing feature.

Selling a property:

- If players decide to sell their property, players should click on their property square. After clicking that square, a pop-up screen will come up with related buttons and information about that square. One of the buttons on that screen will be the "sell" button, if the property satisfies the conditions to be sold.
- If players press on the sell button, the selling screen will come up. In selling the screen, the player will choose the player who is the new owner of the player and the selling price of the property.

Mortgage:

- Any owned property without houses and hotels can be mortgaged.
- To mortgage a property, players should click on their property square. After clicking that square, a pop-up screen will come up with related buttons and information about that square. One of these buttons will be the mortgage button and by pressing that button the property will be mortgaged.
- To lift a mortgage, the owner must pay the bank a certain amount of money. This money is calculated as the amount of mortgage money that the bank gives the owner (half of initial paying price) plus 10% of that money.

Building houses and hotels:

- The requirements needed to build houses and hotels, will be automatically checked by the game.
- If the requirements are met, after pressing on the property on the board, related buttons which are needed to build a house, or a hotel will be visible on the pop-up screen.
- Players can also sell the houses and hotels back to the Bank if all requirements are met. If requirements are met, on the pop-up screen which will be opened after pressing on a property, “sell house” and “sell hotel” buttons will be enabled similar to the “build house” and “build hotel” buttons.
- Information about the houses and hotels such as the number of houses on that property will be shown on the pop-up screen associated with that property square.

Next turn:

- After performing their actions players will declare that their turn is over by pressing the “Next Turn” button and the next user will roll the dice for his/her turn.

3.1.2. Landing on Special Squares

Joker Places:

- If players land on a Joker place, after their pawns are moved to the Joker place, a pop-up screen will come up.

- On the pop-up screen information about the Joker will be given. For example, if the Joker place is a Jail but also require a money penalty, this information will be given in the pop-up screen.
- On the pop-up screen there will be buttons in order to perform required actions. For example if the Joker place requires money penalty, there will be a button “pay” that players will push. After actions are performed , pop-up screens will be closed.

Community Chest and Chance:

- The task on the card will be done automatically by the game.
- Some of the cards will have randomly generated numbers (i.e. the amount of money that will be earned, number of squares that the player will move forward/backward).

3.1.3 End of the game

- Bankrupted players will be detected by the game automatically and be disabled to perform any action.
- The game will end when all the players except one goes bankrupt. The player who doesn't go bankrupt will be the winner.

3.2 Requirements about Editing the Game Features

- If the user chooses the Edit option, they can edit any one of the saved boards, including the default board that the game will provide.
- On the board, all the squares except the default Start square can be edited. There is no limit to how many squares of one type the user can have. In other words, if the user wants, they can make all the squares Properties or they can make all of the Chance squares. It all depends on their preferences.
- Each square must be one of 4 types of squares: Property, Chance, Community Chest and Joker. The features of Property and Joker squares can be edited. Chance and Community Chest squares, on the other hand, can only be added to the board. No further editing can be done on these two types of squares, since the actions on these squares are determined by card decks that cannot be edited, only the currency is replaced.

3.2.1 Editing Properties

- Each property should be given a name, color group and initial buying price.
- All of these features can be edited on the editor. The user can either add the property to an already existing color group or they can create a new color group.
- If they choose to add a new color group, they must select a color from the color chart and determine a name for that specific color group.

3.2.2 Editing Joker Squares

- There are three main features that Joker places offer to players: waiting for a few turns, being moved to few squares forward or backward and get/lose money.
- Players cannot make a Joker square have the properties of both waiting for a few turns and moving forward/backward and this will be checked by the game. Either waiting or moving can be chosen for a square. The third feature, getting/losing money can be used in combination with the other two features or on its own.
- The player can also create a Joker square with no purpose, similar to “Free Parking” in the original Monopoly.
- Extensibility of editing Joker places will be limited by the game. For example, if players create a Joker square to make the player move ahead, there will be a limit to the number of squares they can move such as being able to move at most 5 squares.

3.2.3 Editing pictures

- On certain squares of the board, pictures can be uploaded. These squares include Joker squares and Chance and Community Chest squares.
- Pictures can also be attached to the tokens of the players to create a more custom game.
- Pictures can be selected from the computer and uploaded to the game for each one of the specified squares or tokens.

3.2.4 Editing Miscellaneous Features

- The players can make further customizations on the game like changing the currency or the percentages of house costs or mortgage rates.
- Consequently, all the properties, chance and community chest cards, etc. will be updated regarding the new currency.

3.3 Additional Functional Requirements

- There will be no “Get Out of Jail Free” card in the chance cards since in our Monopoly, with the editing feature, players might prefer not to have any Jail square in the game.
- If a player is in Jail, they can escape from it if they double roll the dice two consecutive times.
- In the board game of the Monopoly, if players double roll their dice three times, they are sent to the Jail. However, in our version this feature will not be implemented since players with the editing feature, players might prefer not to have any Jail square in the game.
- In buying a property feature, the auction option is added. However, the players will have the auction between themselves, and only notify the game only about the last verdict by selecting the new owner of the property and the selling price.
- If a player resigns or goes bankrupt, all of their properties will be put on auction. If there are any buildings on the property, they will be returned back to the bank, only the properties themselves will be auctioned.

4 Non-Functional Requirements

The original limit given to the number of players that can play a single game is 2-8. However, we made the decision to change this limit into 2-4 on our game. Our reasoning is that the UI can get cluttered and confusing with 8 different players on the screen. Also, it can get boring to wait for 7 people to go through their turns especially when playing the game on one computer.

4.1 Usability

- Anyone that knows how to play regular Monopoly and has basic computer skills should be able to play the default game mode. To achieve this, we will try to make the UI resemble the real-life version of the game whenever we can, for example, the board the game will be played in will look like the standard Monopoly board with different themes. In cases we cannot, such as the buttons to choose actions, will be self explanatory with the texts and or icons on them.

4.2 User Friendliness

- It is possible to create many kinds of boards in the Board Maker section. However, these boards might not always be enjoyable or even playable. We will limit the editability of the board to some extent in order to make it harder for the users to accidentally create unplayable boards. For instance, instead of giving the power of creating custom squares to the users, we created different categories of squares which have different editable properties. This will make sure the user cannot select conflicting properties for one square. If the user is given the power of changing every detail of a square, the editing process will take a few hours.
- Moreover, we have limited the number of squares a player can be made to move with a Joker square. That is, a Joker square with the moving property (making a player that lands on it moves a certain number of squares) can move that player at most the length of the board, in order to make the game more playable and to avoid loops.
- We will ask for confirmation when the user tries to exit Board Maker when there are unsaved changes. Also, we will ask for confirmation if the user creates a new board and tries to save it on an existing board. As a final measure, the recently deleted boards will be kept in a separate folder so they can be recovered if needed.

- We added titles on each screen to tell the user where they are. This way they will not get lost between the menus.

4.3 Portability

- We made our boards portable. That is, we created a file system that allows users to drag and drop a “board configurations file” they got from a friend into their “boards” folder. Then, the game should be able to see this board and let the user modify it or play on it. The edited boards will also be saved as portable files.

4.4 Additional Requirements

- In our second iteration we added some user experience features. The most important addition was the dialogs. Dialogs are mainly used to draw players attention to the game. For example pop up dialogs are used to display dice to the screen. Additionally, again by using the dialogs, players will be notified about the square they are on. Otherwise the player could have been confused when he/she jumps 20 squares at once without knowing what happened on the square he/she landed on.
- As editing the whole game from scratch would take a long time, we will be giving a ready to play default board that the users can use as a template for their new board. The default board will have all kinds of squares properties, joker squares, chance, and community chest squares which will give an insight of these squares variety to the user who is editing the board. Also default prices for properties, hotels, houses and rents will be present by default.

5 Pseudo Requirement

- GitHub is used as a version control system in the project.
- Java is the implementation language of the project.

6 System Models

6.1 Use Case Model

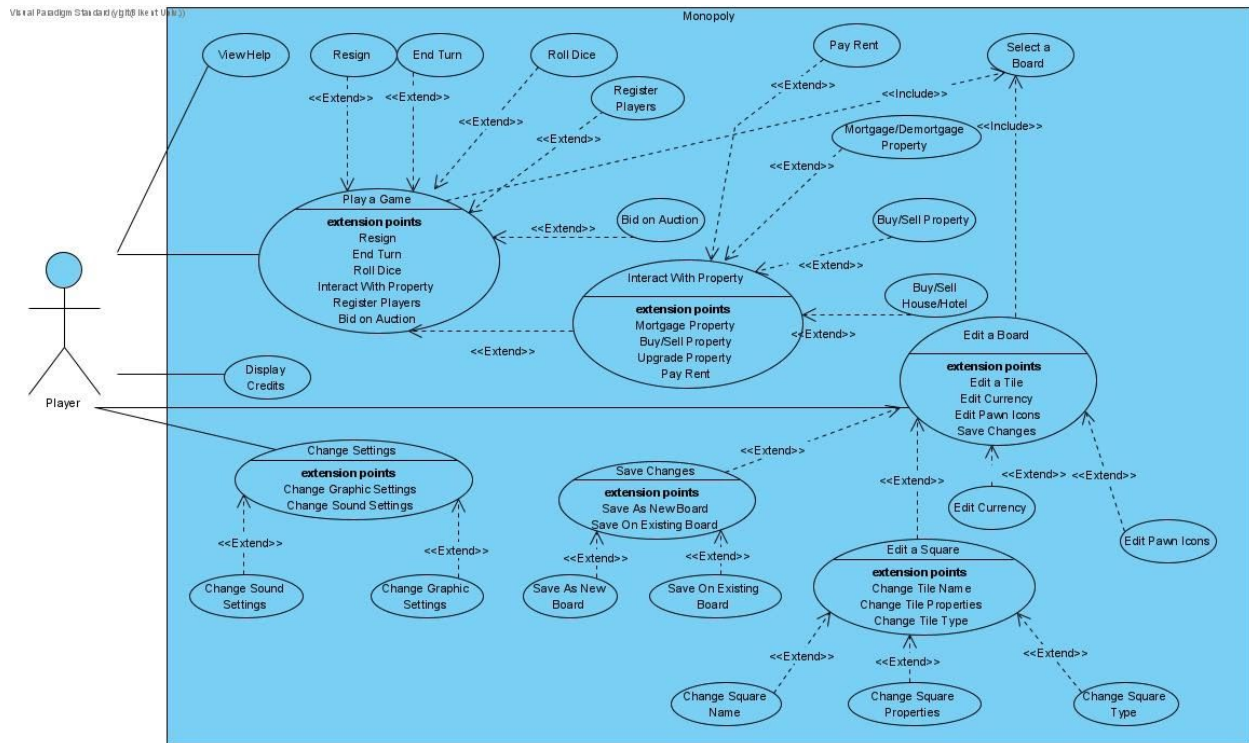


Figure 1. Use Case Diagram

6.1.1 Use Case Name: Play a Game

Participating Actor: Player/Players

Stakeholders and Interests: Player wants to start a new game session.

Pre-Conditions: Player is in the main menu.

Post-Conditions: Player is in the main menu.

Entry-Condition: Player clicks the "Play" button on the main menu, selects a board from the list to play on and registers players.

Exit-Condition: Player clicks the "Exit" button. This can happen during the game session or when the game session ends.

Basic Flow:

1. Player clicks the “play” button on the main menu
2. System displays the available boards to play on as a list
3. Player selects a board from the list
4. All players are registered in player registration screen
5. System initializes a new game session with the settings of selected board
6. Player rolls dice, takes additional actions (these additional actions are the “resign”, and “interact with property” use cases) and ends their turn
7. System processes the dice roll and actions, gives the control to next player in line
8. Players keep taking turns until they go bankrupt or resign (a player who resigns is counted as bankrupt)
9. Only one player is left who is not bankrupt
10. The game session ends, winner is displayed on the screen
11. Player clicks the “Exit” button and returns to main menu

Alternative Flows:

1. During a game, a property is put on auction
 - 1.1. Players bid on the property (this is the “bid on auction” use case)
 - 1.2. “End auction” option is selected
 - 1.3. The highest bidder becomes the new owner of the property and the game is updated accordingly
 - 1.4. Game continues as regular
2. Players want to return to main menu before the game session ends
 - 2.1. Player opens in game menu by pressing “Esc”
 - 2.2. Player clicks the “Exit” button
 - 2.3. System asks for confirmation
 - 2.4. Player clicks “Yes”
 - 2.5. System exits to main menu

6.1.2 Use Case Name: Interact With Property

Participating Actor: Player

Stakeholders and Interests: Player wants to interact with a property

Pre-Conditions: A game session is active, it is the player's turn

Post-Conditions: Game is updated based on interaction

Entry-Condition: Player clicks a property that they own

Exit-Condition: Player clicks somewhere else on the screen

Basic Flow:

1. A game session is active and it is the player's turn
2. Player clicks on a square on the board
3. If the square is interactable (interactable = it is a property owned by the player), a small popup screen containing possible interactions show up
4. Player chooses an interaction (interactions include the "sell property", "buy/sell house/hotel", "mortgage/demortgage" use cases) on the popup screen
5. System processes the interaction and updates the game accordingly
6. Player clicks somewhere else on the screen
7. Popup screen disappears

Alternative Flows:

None

6.1.3 Use Case Name: Edit a Board

Participating Actor: Player

Stakeholders and Interests: The player wants to edit a board or create a new one.

Pre-Conditions: The player is in the main menu.

Post-Conditions: The player is in the main menu.

Entry-Condition: Player clicks the "Board Maker" button on the main menu. User selects a board from the list to edit or clicks the "Create New Board" button.

Exit-Condition: Player clicks the "Exit" button.

Basic Flow:

1. Player clicks the "Board Maker" button on the main menu
2. System displays the available boards to edit as a list
3. Player selects a board from the list to edit
4. System loads and displays the board editor screen with the settings from selected board

5. Player makes changes on the board (making changes include the “edit pawn icons”, “edit currency” and “edit a square” use cases)
6. Player is done with the changes and clicks the “Save” button
7. System asks for confirmation
8. Player clicks “Yes”
9. System saves the changes
10. Player clicks the “Exit” button and returns to the main menu

Alternative Flows:

1. Player wants to create new board instead of editing an existing one
 - 1.1. Player clicks the “Create New Board” button instead of selecting a board from the list
2. Player wants to save edited the board as a new board
 - 2.1. Player clicks the “Save As New” button instead of the “Save” button
 - 2.2. System asks for a name
 - 2.3. Player enters name
 - 2.4. System saves the current board settings as a new board with the given name
3. Player wants to exit without saving
 - 3.1. Player has unsaved changes
 - 3.2. Player clicks the “Exit” button
 - 3.3. System asks for confirmation with the warning message “You have unsaved changes. Do you wish to exit without saving?”
 - 3.4. Player clicks “Yes”
 - 3.5. System exits to main menu

6.1.4 Use Case Name: Edit A Square

Participating Actor: Player

Stakeholders and Interests: The player wants to edit a square on the board.

Pre-Conditions: The player is in the editing menu.

Post-Conditions: Changes are processed and the board is updated accordingly. Player is still in the editing menu.

Entry-Condition: Player clicks on a square while in the editing menu.

Exit-Condition: Player clicks “done” or “back”

Basic Flow:

1. Player is in the editing menu
2. Player clicks on a square on the board
3. Options to change the properties of the clicked square are displayed on a small popup
4. Player makes changes on the square by interacting with the popup (these changes include the “change square properties”, “change square name” and “change square type” use cases)
5. Player clicks “done”
6. Popup disappears
7. Changes are saved and board is updated accordingly

Alternative Flows:

1. Instead of clicking “done” on step 5, the player clicks “back” instead
 - 1.1. Popup disappears
 - 1.2. Changes are not saved

6.1.5 Use Case Name: View Help

Participating Actor: Player

Stakeholders and Interests: The player wants to learn how to play or how to edit boards.

Pre-Conditions: The player is in the main menu or the in-game menu.

Post-Conditions: The player is back at the screen before they view help.

Entry-Condition: Player clicks the “Help” button on the main menu or the in-game menu.

Exit-Condition: Player clicks the “Back” button.

Basic Flow:

1. Player clicks the “View Help” button on the main menu
2. System loads the “Help” screen where information about the game is displayed in text
3. Player reads the text and acquires required information
4. Player clicks the “Back” button

5. System exits to main menu

Alternative Flows:

1. Player clicks the “View Help” button in the in-game menu instead of the main menu
 - 1.1. When Player clicks the “Back” button, system returns to the screen the “View Help” button was clicked instead of going directly back to main menu

6.1.6 Use Case Name: Change Settings

Participating Actor: Player

Stakeholders and Interests: The player wants to learn how to play or how to edit boards.

Pre-Conditions: The player is in the main menu

Post-Conditions: The player is back at the main menu

Entry-Condition: Player clicks the “Settings” button on the main menu

Exit-Condition: Player clicks the “Back” button.

Basic Flow:

1. Player clicks the “Settings” button on the main menu
2. System displays settings menu on screen
3. Player changes the graphics settings (full screen mode and resolution)
4. Player changes the sound settings (mute/unmute)
5. Player clicks the “Back” button
6. System asks for confirmation if the player wants to keep the new settings
7. Player clicks “Yes”
8. System saves the changed settings permanently
9. System exits to main menu

Alternative Flows:

1. Player wants to return to old settings instead of saving the new ones when exiting settings
 - 1.1. When the system asks for confirmation, player clicks “No” instead of “Yes”

- 1.2. System returns to the previous settings that existed before the “Change Settings” button was clicked

6.1.7 Use Case Name: Display Credits

Participating Actor: Player

Stakeholders and Interests: The player wants to learn about the developers.

Pre-Conditions: The player is in the main menu.

Post-Conditions: The player is in the main menu.

Entry-Condition: Player clicks the “Credits” button on the main menu.

Exit-Condition: Player clicks the “Back” button.

Basic Flow:

1. The player clicks the “Credits” button on the main menu
2. System displays the information about developers on the screen
3. The player clicks the “Back” button
4. System exits to main menu

Alternative Flows:

None.

6.2 Dynamic Models

6.2.1 Sequence Diagrams

6.2.1.1 Editing the Game : Properties

Scenario : In this scenario, the player prefers to edit the board given in the game. In order to obtain this, the player presses the edit button in the main menu and a message is sent to Editor class. Following that, the editor class sends a message to Board class to access the selected square. In this scenario, the player decides to make the selected square of type property. After the square becomes a property square, the player edits certain features of the Property square. In the diagram given below, the player prefers to edit the buying price and a name of the property. However, the player can also edit other attributes of a property such as its rental price.

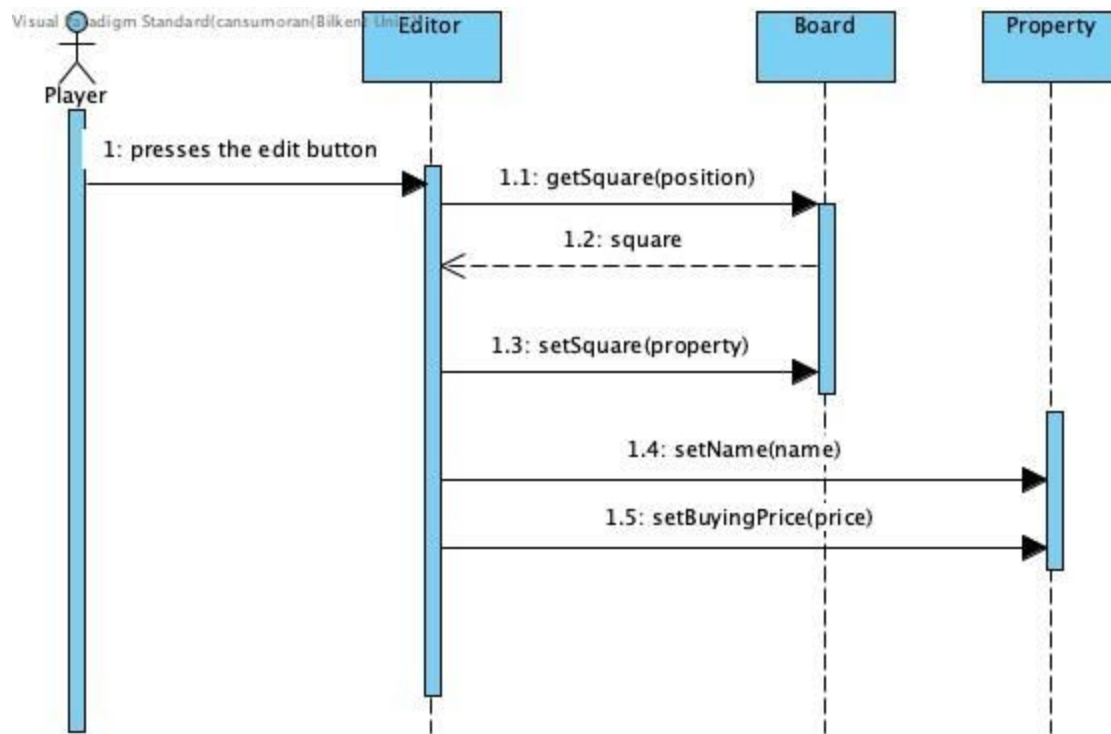


Figure 2. The sequence diagram regarding the editing process of a property

6.2.1.2 Landing on a Property and Buying it

Scenario : During the game, the player rolls the dice when it's his/her turn to play. Based on the result, the player's pawn will be moved to the square which is the "result" ahead of the player's current square. After the pawn is moved, the current square will be checked from Board class. Assuming the square is a property square which hasn't been bought yet, the player will be able to buy it. Following that, the property's current owner will be updated as the player by Property class and player's money will be updated by the message that Game Manager sends to the Player lifeline. At last, the player will press the next turn button which will send a message to GameManager in order to continue the game.

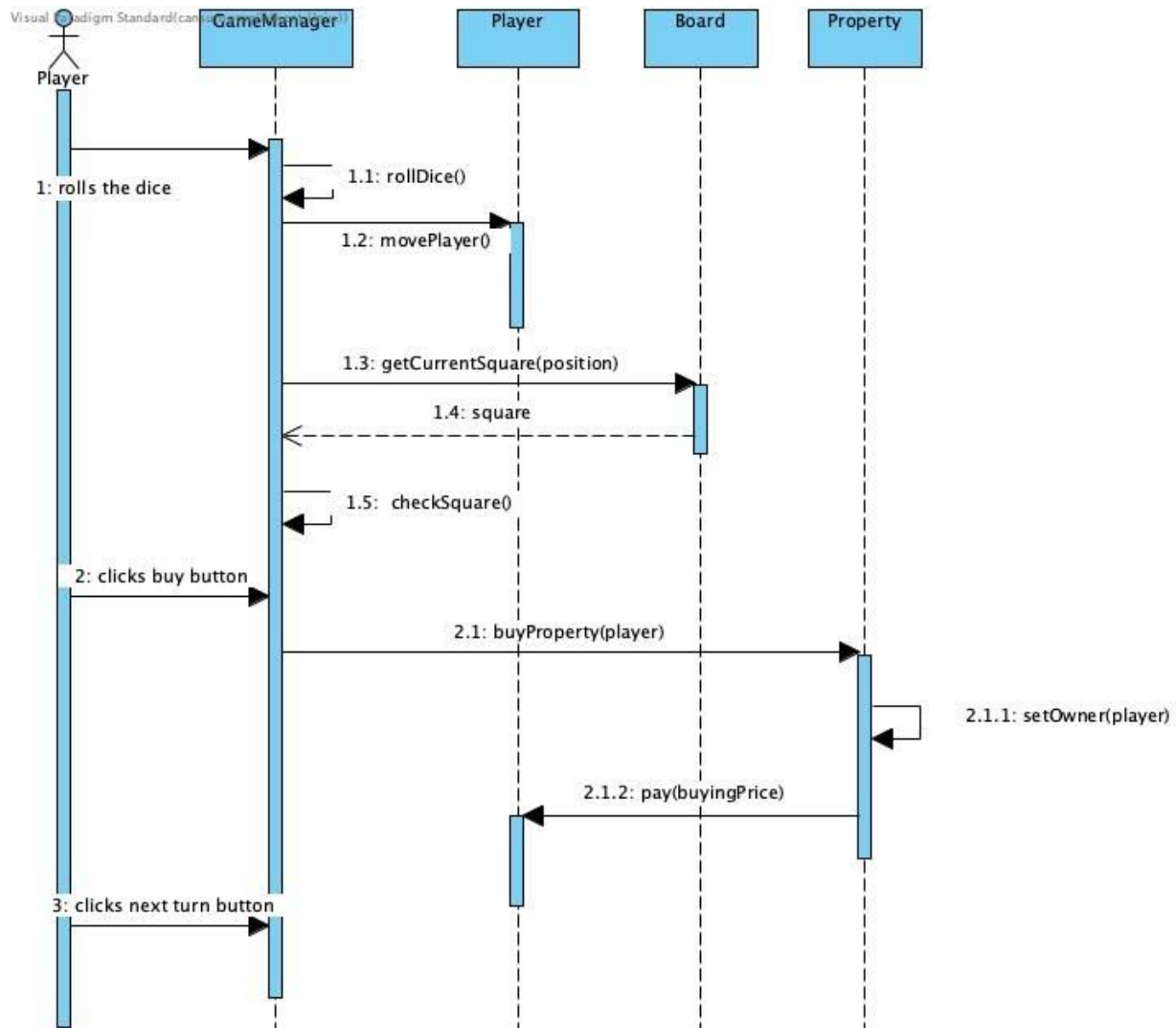


Figure 3. The sequence diagram regarding landing on a property and buying it

6.2.1.3 Landing on a Joker Square and Paying Rent

Scenario : In this scenario, the player rolls the dice and based on the result that they get, the player's pawn is moved to a Joker place by GameManager. After getting the square information from the Board class, the features of the landed Joker square is implemented. The Joker place that the player landed on has the features making the player move a few squares ahead and make them earn money as prize. Joker class returns the money prize to GameManager and Game Manager sends a message to Player class in order to update player's money. Following that, Joker place returns the amount of squares that the player's pawn will be moved ahead to the GameManager. The final square that the player's pawn lands on is a property, which is

learnt from the Board class. The property that the player lands on in this scenario is owned by other players. After getting the rent and owner of that property, the GameManager takes rent from the current player and pays the given amount of money to the owner of that property using pay and gain functions of the Player class. At last, the player presses the next turn button in order to continue the game.

6.2.1.4 Landing on Community Chest and Chance Squares

Scenario : In this scenario, first the player lands on Community Chest square then, lands on Chance square. First, the player rolls the dice and according to the result, the player's pawn is moved to a Community Chest square by the Game Manager. After getting the square information from the Board class, Community Chest card is drawn and Card lifeline receives a message to return the action required on the drawn card to Game Manager. According to the drawn card, the Game Manager needs to move the player's pawn a few squares back, however, after the pawn is moved back, it lands on a Chance square. Chance card is drawn again by the Game Manager and it gives money to the player as a prize. At last, player's money is updated according to the prize by the message that Game Manager sends to the Player class. Then, the player clicks the next turn button in order to continue the game.

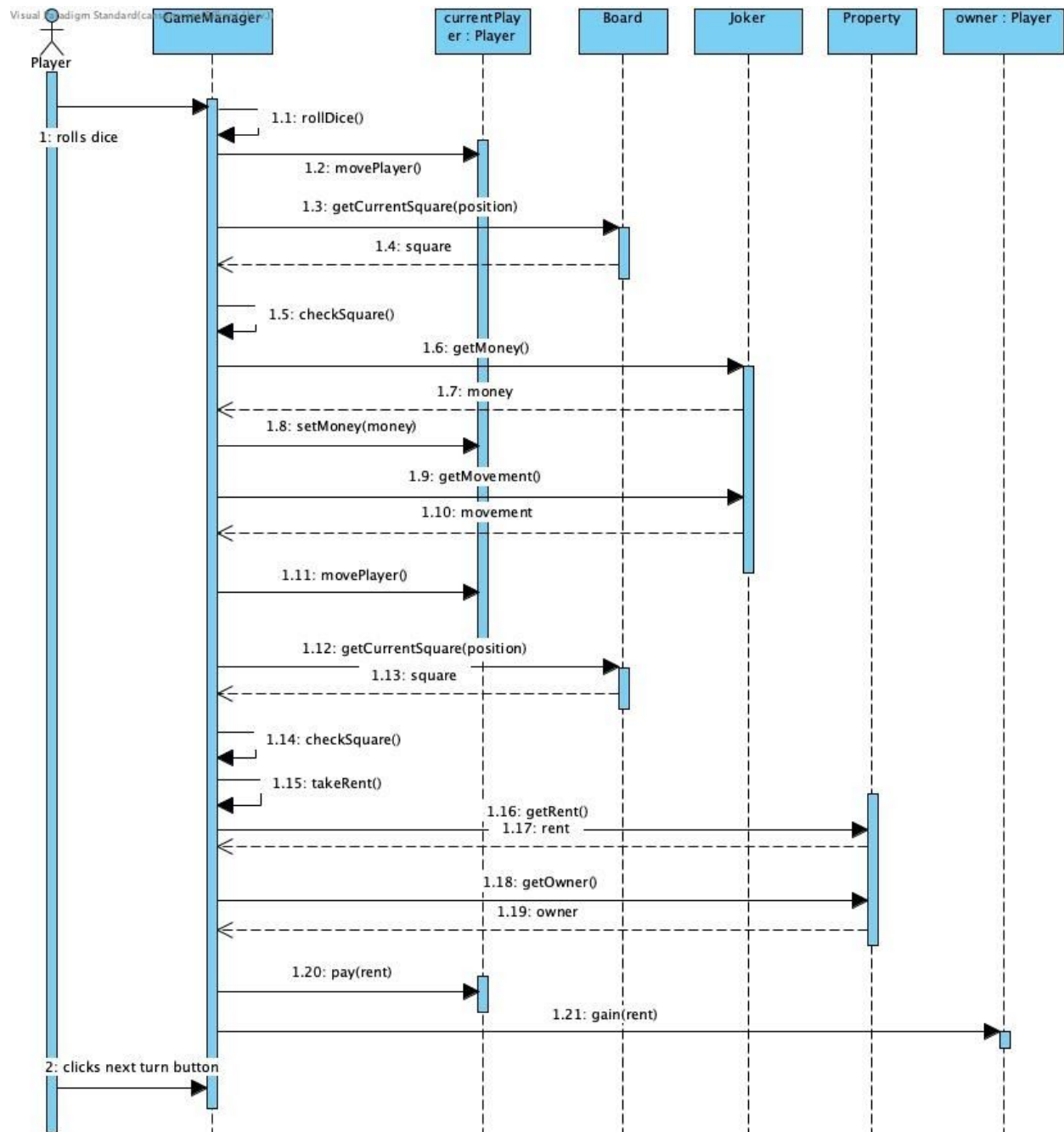


Figure 4. The sequence diagram regarding landing on a joker square and then paying rent

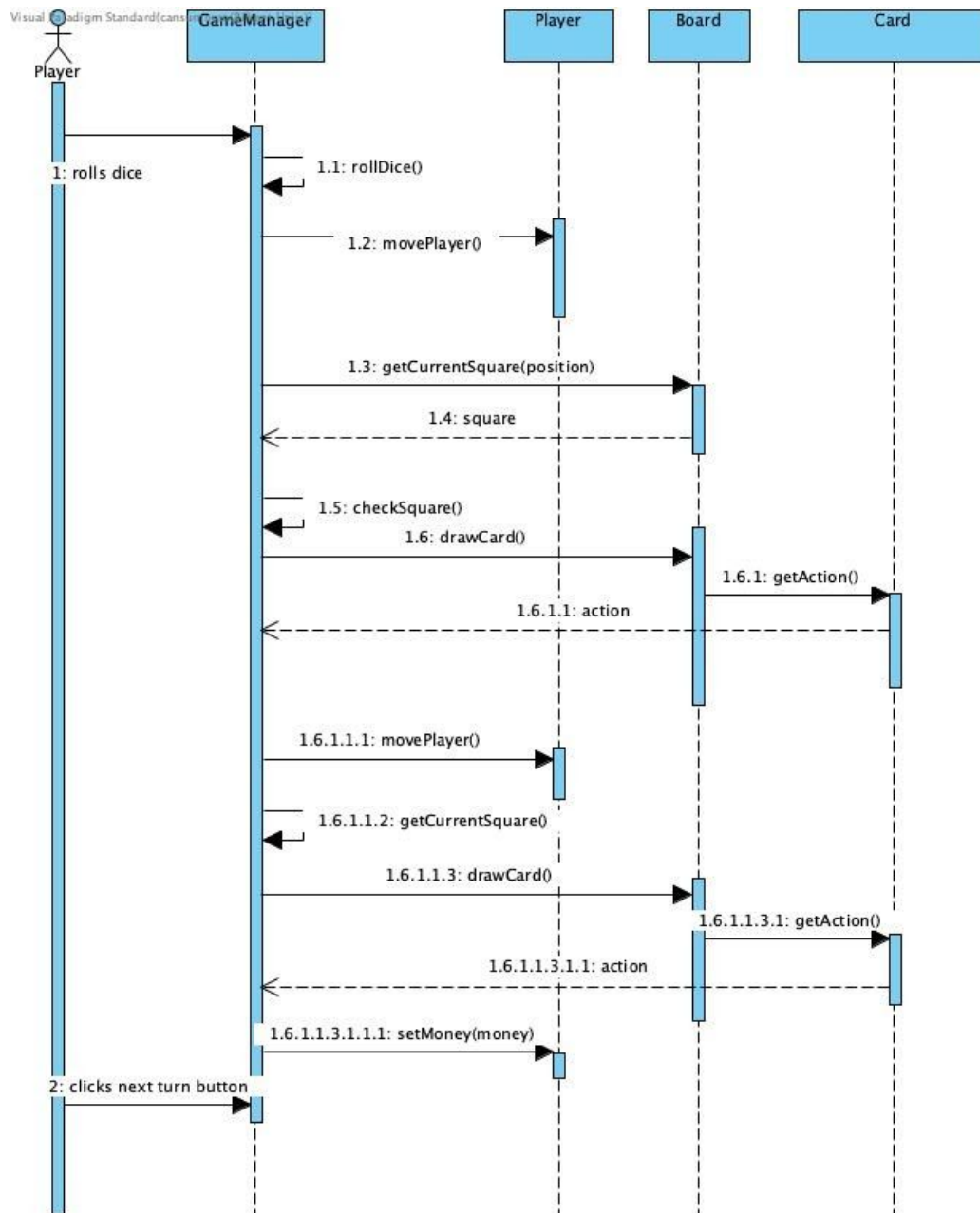


Figure 5. The sequence diagram regarding landing on community chest and chance squares

Figure 6.1 The first part of the activity diagram: game play



Figure 6.1 The first part of the activity diagram: game play

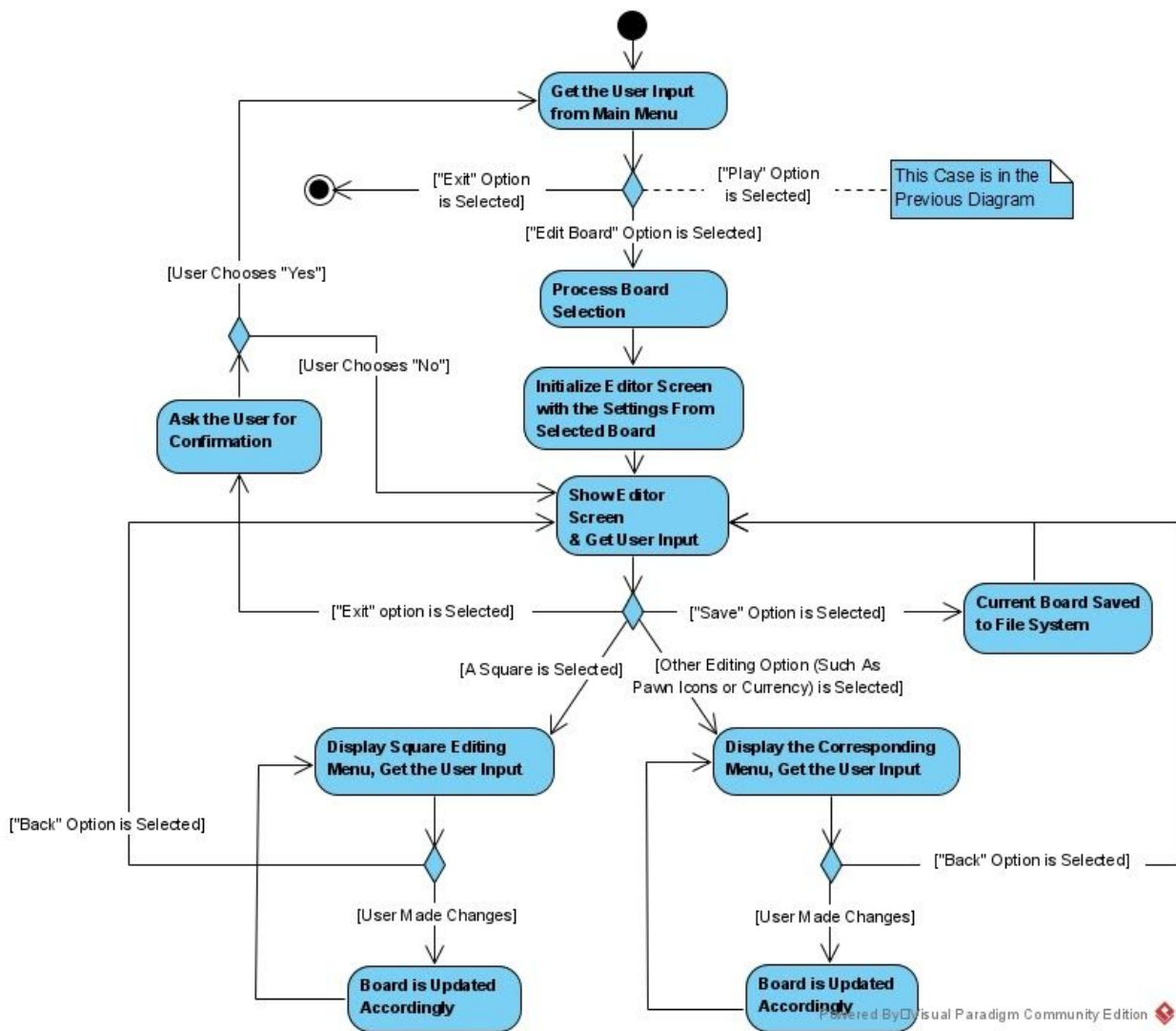


Figure 6.2 The second part of the activity diagram: editing

To get the activity diagram to both fit into the report and still be readable, we split the diagram into two. At the start, the system is at the main menu waiting for user input. In the main menu, the user can choose to exit, play a game or edit a board (other minor and uninteresting options such as displaying credits are omitted in the diagram). The first part of the activity diagram deals with the game play scenario, whereas the second part deals with board editing. If the user selects exit, the program simply terminates.

6.2.2.1 Play a game

If the user selected “play a game” in the main menu, a board selection menu will appear displaying available boards. Then the user can select a board to play on and enter the names of players. Then the system initializes a game session with the given player names and selected board settings. After that the following game loop (which symbolizes a turn) keeps happening until there is only one player left that is not bankrupt.

System selects the next player in line. If the player is not in jail (i.e. does not have positive jail time), they roll the dice. Note that we will mention the jail case in the next page. System calculates the player's next position based on the dice roll and processes the requirements of the newly entered square. This possibly includes changing the player's funds and drawing a chance card. If the new square or the drawn chance card requires movement, a new location is calculated and this process repeats. If there is no requirement to move again we continue. Here, we check if this location has a “suspend” property (i.e. functions like a jail). If it does have this property, we suspend the player there for the specified number of turns. If not, we continue. At this point, the player is standing on a square with one of these four cases:

Case 1: Square is not a property. → Nothing happens.

Case 2: Square is a property and is owned by the player themselves. → Nothing happens.

Case 3: Square is a property and is owned by another player. → The necessary money transaction for rent is done, and the balances of the owner and the current player are updated accordingly.

Case 4: Square is a property and is not owned by anyone. → First, the player is asked if they want to buy the square. If the answer is yes, the square is now owned by the player and their funds are adjusted accordingly. The board gets updated based on the new owner. If they did not want to buy, then the square was put on auction. If someone buys it in this auction, their funds are adjusted accordingly and the board gets updated based on the new owner. The square is sold to the highest bidder. All players can bid, including the current player that declined the square in the beginning. Bidding can begin at any price. If, however, the square is not purchased by anyone in the auction, the system simply continues.

We ask the user if they want to perform additional actions. This includes selling, upgrading and mortgaging a square that they own. The player is allowed to perform as many of these actions

as they wish. We move onto the next step after they do all they wish and finally choose to end their turn.

Here is a good place to mention what happens to a player that was in “jail” in the beginning of the turn. We ask such a player if they want to roll dice, which comes with a specified fine. If the player does not want to roll, we simply decrease their remaining jail time by 1 and move on to the step of asking them if they want to perform additional actions such as mortgaging. This is legitimate based on the official rulebook. On the other hand, if the player did want to roll, we subtract the jail roll fine from their balance and make them roll dice. If they manage to roll double, their jail time drops down to zero, they become free and can roll dice again to continue as a regular player. If they cannot roll double, we again just decrease their remaining jail time by 1 and move on to the step of asking them if they want to perform additional actions such as mortgaging.

We should note that, since we do not have a specific jail square, we have omitted the following two features: “get out of jail card” and the “rolling double three times in a row puts a player in jail.” We do not have a specified jail square but instead have joker squares that can act as jails. As the number of these can be zero or more than one, the mentioned two features would have been somewhat arbitrary.

In both the “player is in jail” and “player is not in jail” case, we ended up at the step of asking the player if they want to perform additional actions. Here, the player can perform such an action as many times as they can and want, or they can choose to end their turn, or they can choose to resign from the game completely. If they choose to resign, the system disables them, pays their debts to the other players and the bank, and puts their properties (except for buildings) on auction one by one. Otherwise, possibly after performing additional actions, if the player chooses to just end their turn, we check the last three dice rolls of the player to see if they had rolled double three times in a row. If so, the player’s turn is ended automatically, to prevent them from progressing too much (note that this is our changed version of the feature “put the player in jail after three double rolls”). If they had not rolled double in the last roll, again their turn just ended. If, however, the player rolled double the last time and rolled double maximum twice in a row, they are asked to roll dice again, and repeat the game loop.

When a player's turn eventually ends, we check if they are bankrupt (if their balance is negative). If so, we simultaneously disable that player and pay their debts to the other players and the bank, after which we put all their properties (apart from buildings) on auction one by one. If they are not bankrupt, we just skip that step. Finally, we check if all players but one are bankrupt. If so, the game ends and the system goes back to the main menu after showing the game results in the game end screen. If not, we simply pick the next player in line and go to the beginning of the game loop.

6.2.2.2 Edit a Board

If the user selected the "Edit a Board" option in the main menu, a board selection menu will appear displaying available boards. The system processes their board selection. Here, the user can select a board to edit or "create a new board". Creating a new board will create a copy of the default board for the player to edit. The system initializes the board editing menu based on the settings of the selected board. Then the player can keep making changes to the board until the following loop ends:

System gets user input. There are four cases:

Case 1: The player selected a square. The menu to edit that square's settings appears. The user edits some of those settings. The changes are applied and the board is updated accordingly. System goes back to waiting for input. If at some point the user presses "Back" instead of making a change, the system again goes back to waiting for input.

Case 2: The player selected some other setting (like the name of the currency or chance deck). The menu to edit that specific setting appears if it exists. No menu might be needed for simple edits like changing the name of the currency. The user edits the desired setting. The changes are applied and the board is updated accordingly. System goes back to waiting for input. If at some point the user presses "Back" instead of making a change, the system again goes back to waiting for input.

Case 3: The input is to save. Then the current board is saved into the file system, and the system goes back to waiting for input.

Case 4: The input is to exit. Then the loop ends and the system opens the main menu.

6.2.3 State Diagrams

These state diagrams track the states of interesting entities during an active game session. Outside the game session, objects rarely behave differently based on their current state, therefore we chose to focus only on the game session.

6.2.3.1 Player State Diagram

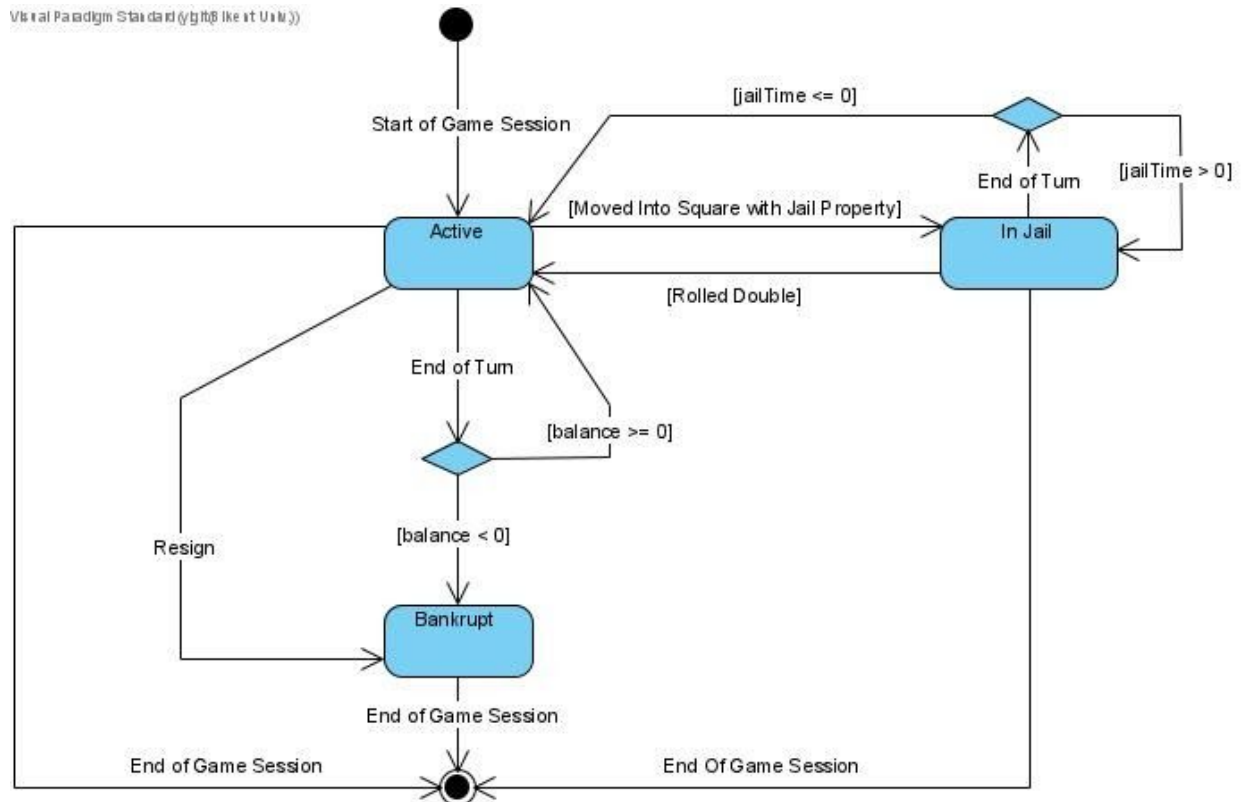


Figure 7. The Player State Diagram

At the start of a game session, a Player object is initialized with the appropriate values and enters the Active state. In this state, players can do regular actions like selling/improving properties and default processes are applied when dice is rolled. These actions can modify the player's balance.

When a player ends up in a square with jail property, they are suspended for a given number of turns. Everytime they end their turn their jail time is decremented and also they can pay a fee to

roll dice each turn. If they roll double, they go back to Active state and can roll again and proceed like normally. When their jail time is 0, Player moves back to Active state.

“End of Turn” button is disabled by the Game if the player has not rolled dice yet, therefore the player object itself does not need to check for dice roll condition. It will never receive an “End of Turn” call unless the player has already rolled dice that turn.

The turns of players are also managed by the Game so the Player object itself does not need to keep track if it is this (by this it is meant the Player object we are tracking the state of) player’s turn or not. No matter what state the player is in, it will never receive an “End of Turn” call unless it is this player’s turn.

If the Player resigns or ends a turn while having negative balance the player will be bankrupt. In this state, the Player object still exists and its final properties before going bankrupt are unchanged so they can be displayed at the end of the game session. However the Game will ignore bankrupt Players when calculating turns and other processes. When a player moves to the bankrupt state, the Game puts all properties owned by that player to auction one by one.

6.2.3.1.1 Property State Diagram

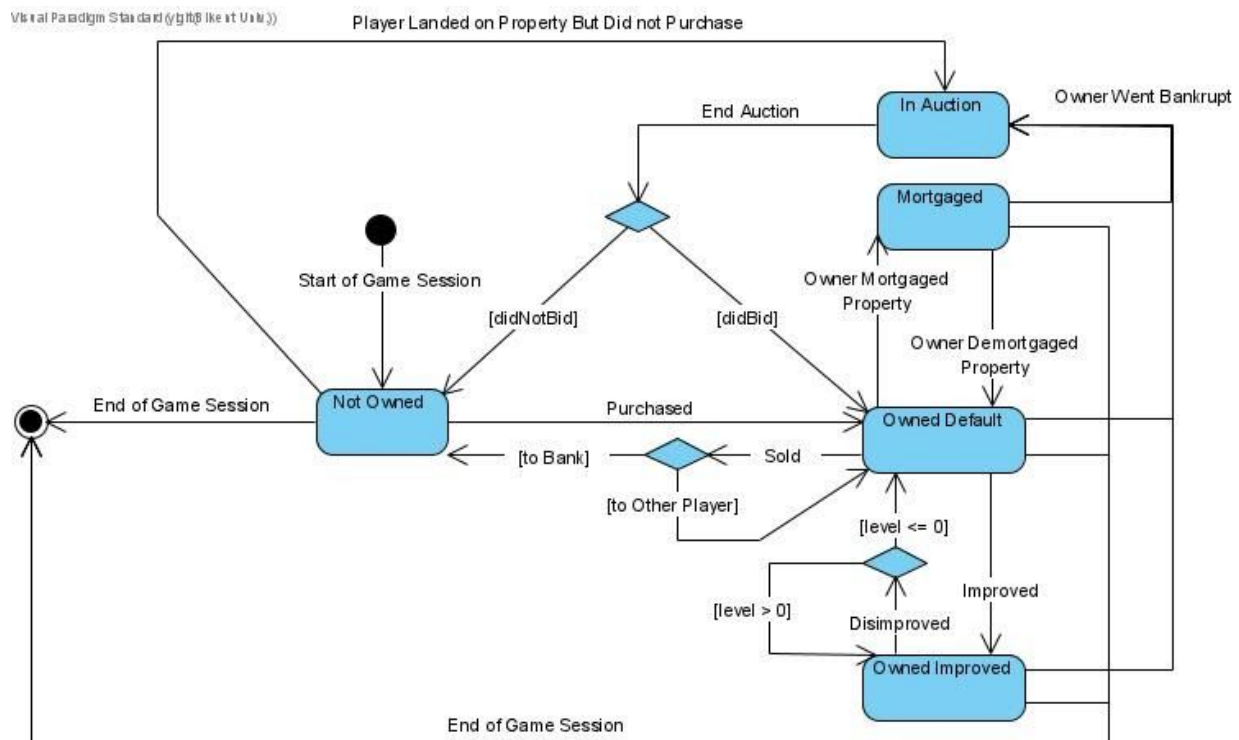


Figure 8. The Property State Diagram

During the initialization, Properties are created without an owner (here, owner would be a reference to a Player object). Properties with no owner are represented by the “Not Owned” state. If during the game, a Player “purchases” a Property, the owner of the property is set to the purchaser and Property moves to the “Owned Default” state. This state represents a newly bought, unimproved, not-mortgaged Property.

“Purchases” can happen in 3 ways:

- A Player lands on a “Not Owned” property and purchases it.
- A Player lands on a “Not Owned” property and does not purchase it. Then, the property is put on auction. The Player with the highest bid at the end of an auction purchases it.
- The owner of the property goes bankrupt. Then, the property is put on auction. The Player with the highest bid at the end of an auction purchases it.

A property in “Owned Default” state can be sold. If the Property was sold to the bank, it will move to “Not Owned” state. If the Property is sold to another Player, its owner is changed accordingly and it goes back to “Owned Default” state.

A Property in “Owned Default” or “Owned Improved” state can be improved under the following four conditions:

- The “Color Group” the Property belongs to is “complete”. Complete means all Properties in the group are owned by the same Player.
- The improvement is “balanced”. That is, the Property must not have a higher improvement level than any other property in its “Color Group”.
- The Property must have an improvement level less than the “Maximum Improvement Level”.
- The none of the Properties in the “Color Group” are “Mortgaged”

If all four conditions are satisfied the property can be improved (its improvement level is incremented). If a Property in “Owned Default” state is improved, it moves to “Owned Improved” state, in which case it can no longer be mortgaged or sold.

A property in “Owned Improved” state can be disimproved (its improvement level is decremented). If the improvement level reaches “0” after disimprovement, the Property moves to the “Owned Default” state. The disimprovement must also be balanced. That is, the improvement level of the Property must not be less than any other Property in the “Color Group”.

A Property in “Owned Default” state can be mortgaged, in which case it moves to “Mortgaged” state. A Property in “Mortgaged” state can be “demortgaged”, in which case it moves back to “Owned Default” state. To see more details on how mortgage rates are processed, please check out Activity and Class diagrams.

6.2.3.1.2 Color Group State Diagram

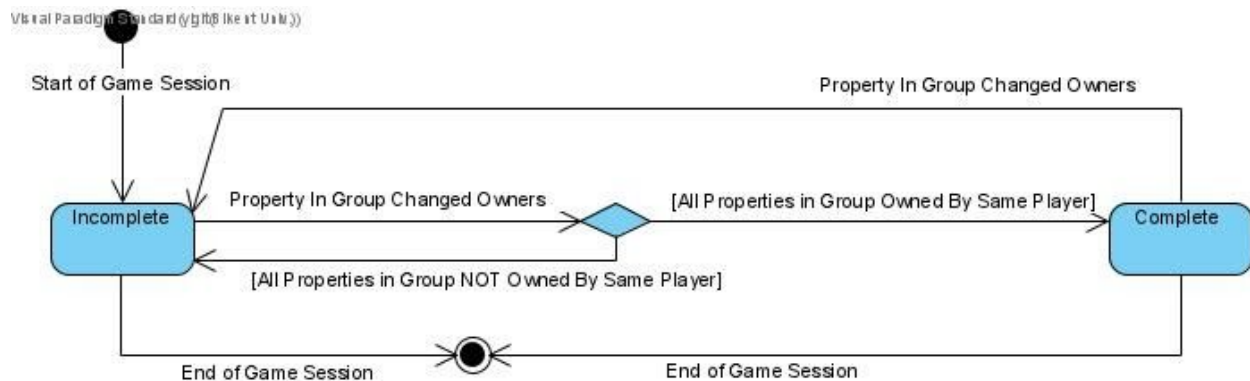


Figure 9. The Color Group State Diagram

A Color Group holds references to an arbitrary number of Properties in a list. Based on changes in any of these Properties, the state of the Color Group is updated.

If the Color Group is “Incomplete” and an update is called through a change in one of its Properties, there are two possibilities:

- If all the Properties’ of the Color Group belong to the same Player, the Color Group moves to “Complete” state.
- If If all the Properties’ of the Color Group DO NOT belong to the same Player, the Color Group moves back to “Incomplete” state.

If such an update call occurs while the Color Group is in “Complete” state, we know that no matter who the new owner of the modified Property is, it is impossible that all Properties have the same owner. Therefore the Color Group goes directly back to the “Incomplete” state.

Color group is mainly used to determine if a Property is improvable and or disimprovable. Details of how this is determined are covered in the explanation of Property State Diagram. The important property of a Color Group is that it provides the knowledge if a Property’s Color Group is “complete” or not. As mentioned before, a property can only be improved if its color group is “complete”.

[illegible]

Screen: Screen class is the class which will handle the contents of a screen that needs to be drawn.

BoardSelectionScreen: BoardSelectionScreen will handle showing the saved boards on screen as well as getting the user input of which board to open for edit or play options.

PlayerSelectionScreen: PlayerSelectionScreen acts as the user interface of the PlayerManager. It will draw the player options on screen.

GameScreen: GameScreen acts as the user interface of the GameManager. It will handle drawing and updating the board view on screen as well as all the dialogs that will come up according to the square that the players land on. It will be responsible for updating the game view.

EditorScreen: EditorScreen acts as the user interface of the EditorManager. It will handle getting all the user input from the screen about editing the board features.

ScreenManager: ScreenManager is the class that handles all the screen activities. It controls the switches between different screens.

EditorManager : EditorManager class handles all the editing actions performed. It gets the user inputs from the EditorManagerScreen and changes the Board that is being edited accordingly. It handles all the logic behind editing.

GameManager: GameManager class is defined for the game control of the game. It will be the general controller of the game by handling most of the features in game play and controlling interactions between many classes. Furthermore, it will provide functions for buying/selling houses and properties, as well as controlling player turns.

BoardManager: BoardManager class is defined for managing Monopoly boards in the game. In the game, players will be able to edit the given game template. Furthermore, players will be able to save their edited templates on a page. This class will provide functions for naming these edited templates as well as selecting them from the page to play.

FileManager: FileManager class will be the class which interacts with the local file system in the computer, and will handle the functions which are needed to read and write the board data to the local file system.

JSONable: JSONable is an interface which requires the classes that implements it to be capable of two tasks:

- Be able to create a JSONObject that contains its current state and return it
- Extract properties from a JSONObject that was created by an instance of the current class

This allows us to get the states of entities in JSON format easily and write them to the filesystem using the filemanager. Also, we can read a previously stored JSON from the filesystem and reconstruct the entity states with it.

PlayerManager: PlayerManager class is defined to manage player selections including adding a player to the game, deleting a player, selecting player colors and selecting the token of a player.

Board: Board class is defined for the game board in the Monopoly game. It contains squares in addition to pawns selected for the game. It also has chance and community chest card decks.

Player: Player class is defined for the players in the game. It will provide functions for setting a pawn to the player. Furthermore, it will provide information about the player. For example, it will keep the name, the amount of money that player has, associated color of a player and the player's current position on the board. It will also check whether the player is bankrupt or not.

Pawn: Pawn class is defined for tokens or in other words, pawns in the Monopoly game. It includes the name of the pictures selected from the computer.

Square: Square interface is defined for the squares in the Monopoly board. In our Monopoly, there are four types of squares which are Start, Joker, Property and ChanceandCommunity that each implement the Square interface.

SquareType : SquareType is an enumeration for the square types in our game. The square types include START, JOKER, CHANCEANDCOMMUNITYCHEST, and PROPERTY.

ColorGroup: ColorGroup class is a class which is defined for the “color-grouped properties” in Monopoly. There will be 8 default color groups for properties in the game like in the Monopoly board game. However, players will be able to change the number of color groups, the number of properties inside color groups or the color of these groups and ColorGroup class will provide functions for these changes.

Property: Property class is defined for the Property squares in the Monopoly game. It stores property information including the buying price, rent and color group of that property. For example, if players prefer to change the features of a property such as its name, price, rent price, house price and hotel price, Property class will provide the required functions and change its attributes accordingly. Any in game actions concerning the properties are also carried out through the Property class, such as mortgage property, buy property and add house.

Joker: Joker class is defined for the Joker places in the game. If players prefer to edit the game, they can change the actions in Joker places. For example, if a Joker Place requires waiting for 3 turns, players can change that action into moving 3 places ahead and Joker class will provide required functions for editing Joker places.

Start: Start class represents the starting square of the Monopoly game which is another different kind of a square.

ChanceandCommunity: ChanceandCommunity class is defined for the chance and community chest squares in the game. The only editable feature of these squares is the background picture of them. Therefore, this class will only provide functions for editing pictures for editing purposes.

CardDeck: CardDeck class is defined for the Chance and Community Chest card decks in the Monopoly game and it will provide functions for generating these decks.

Card: Card class is defined for Chance and Community Chest cards in the Monopoly game. It has a strong aggregation relationship with CardDeck class and it will keep the prompt and required action for a card.

SquareGenerator: SquareGenerator class which will handle the required functions in order to create squares based on the stored board information.

6.4 User Interface

6.4.1 Navigation Path

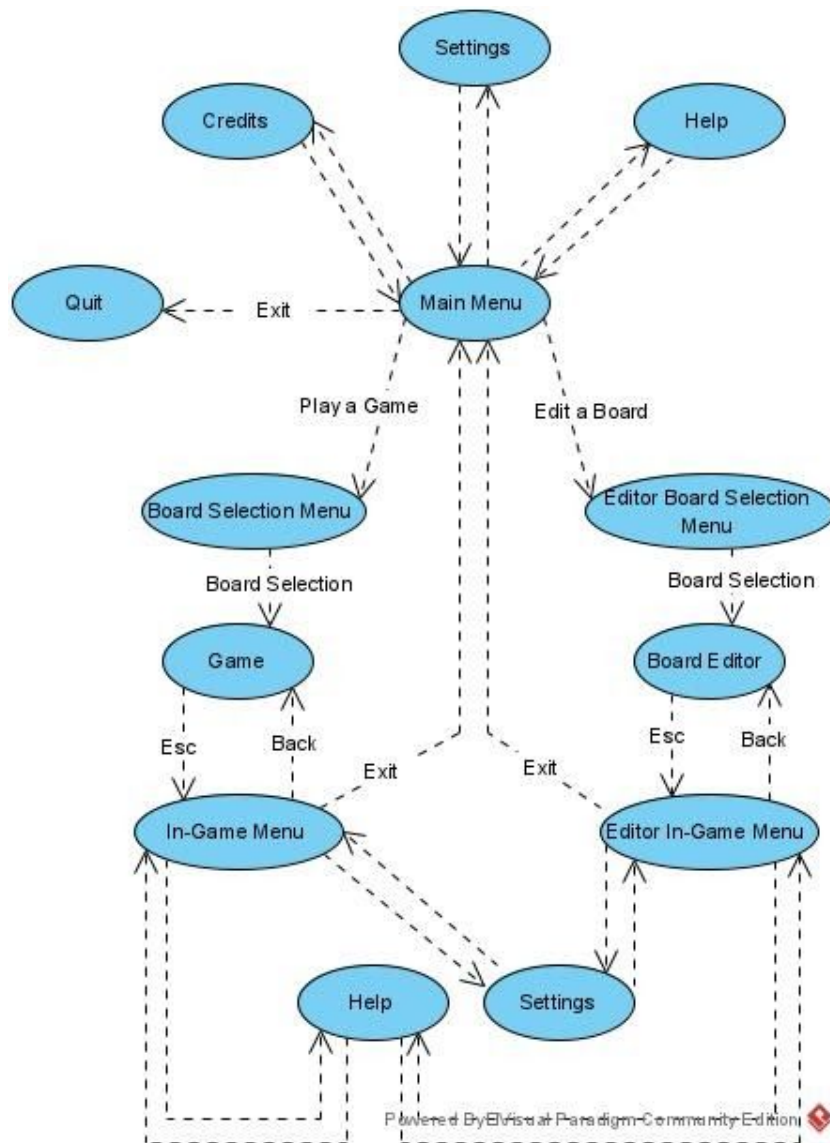


Figure 11. The navigation path of the game

6.4.2 Screen Mockups

6.4.2.1 Main Menu

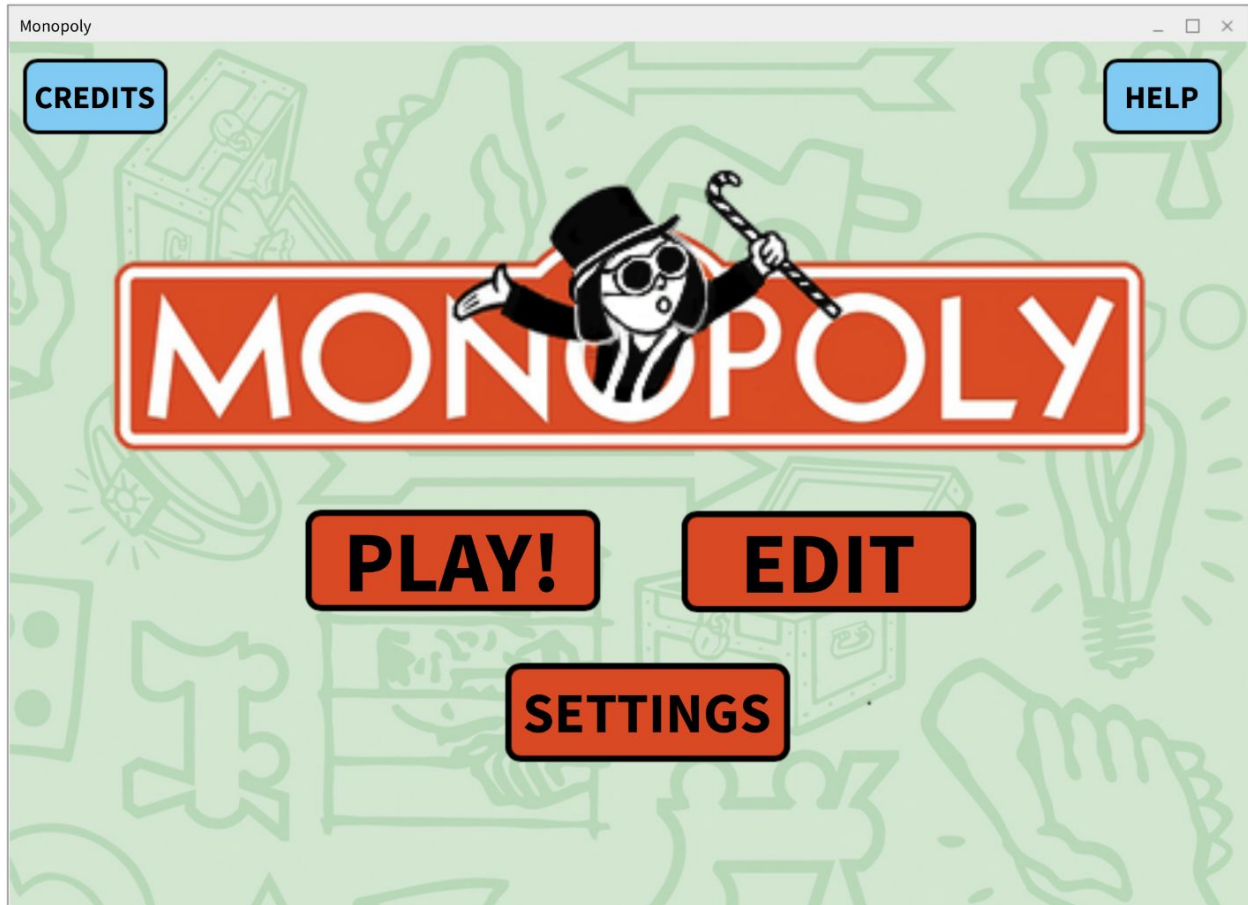


Figure 12. Main menu screen mockup

This is the main menu screen of our Monopoly. It is also the first screen the players will come across with when they open the game. From this window, players can either play the game directly by pressing the “Play!” button, or they can move on to the edit screen by pressing the “Edit” button. Apart from these, there are also 3 smaller buttons named “Settings”, “Credits”, and “Help”. By pressing the settings button, players will reach the setting screen where they can adjust game volume or enter or exit fullscreen mode. The credits button will open the credits screen where users can find information about the game and its developers, us. Finally, the help button will take the users to a manual text about how to play this Monopoly game.

6.4.2.2 Player Selection Screen

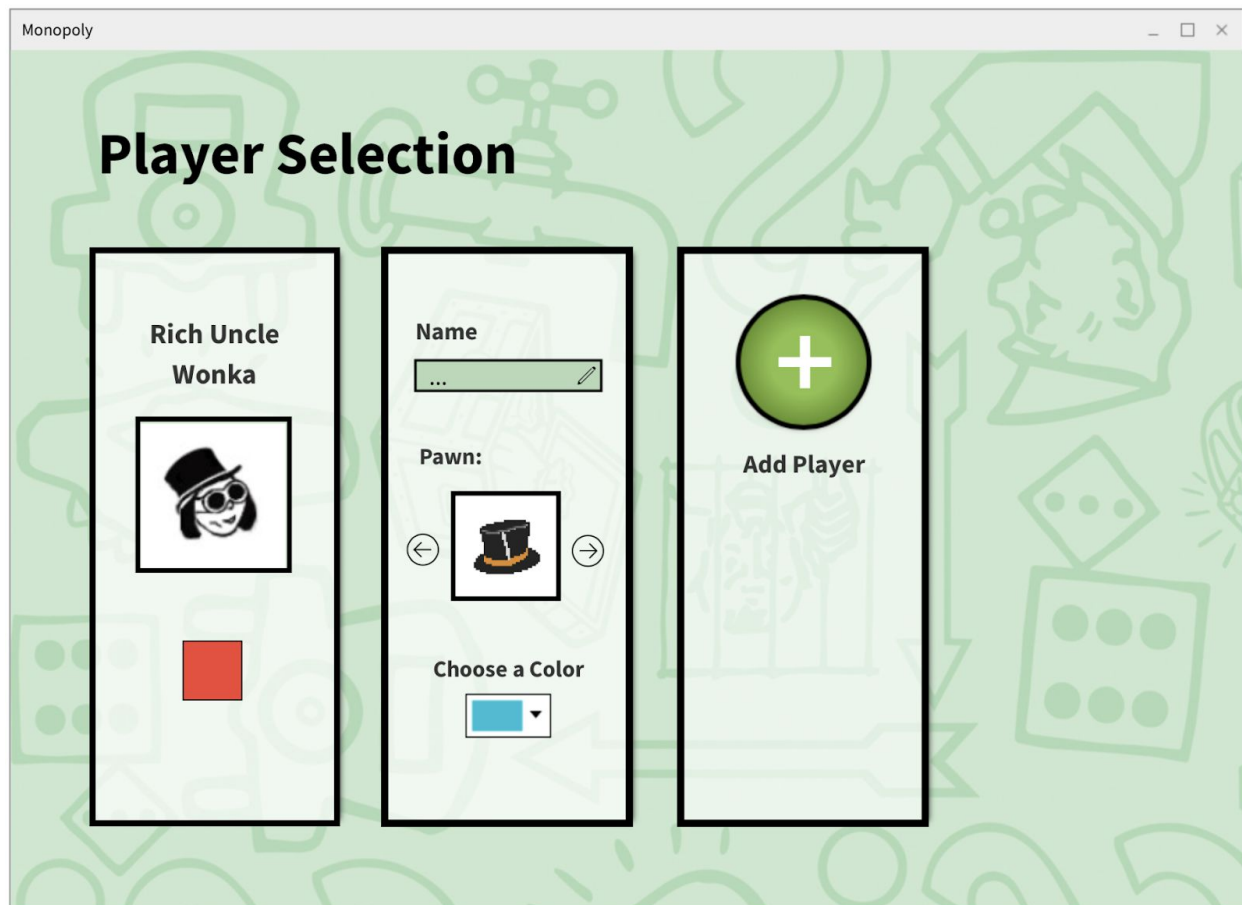


Figure 13. Player selection screen mockup

In this screen, the players of the game will be added. By pressing the green add player button users will be able to add up to 4 players. The leftmost subscreen shows the profile of a player named “Rich Uncle Wonka”. The red rectangle represents the color the player chooses which will indicate his/her properties on the game board.

The subscreen in the middle shows an unfinished profile. The player will enter his/her name on the text box under the “Name” title. After that, users can set their pawns by choosing preset pawns by browsing them using the arrows located on both sides of the pawn picture. Also players will choose a color that will be displayed on the properties they own.

6.4.2.3 Board Selection Menu

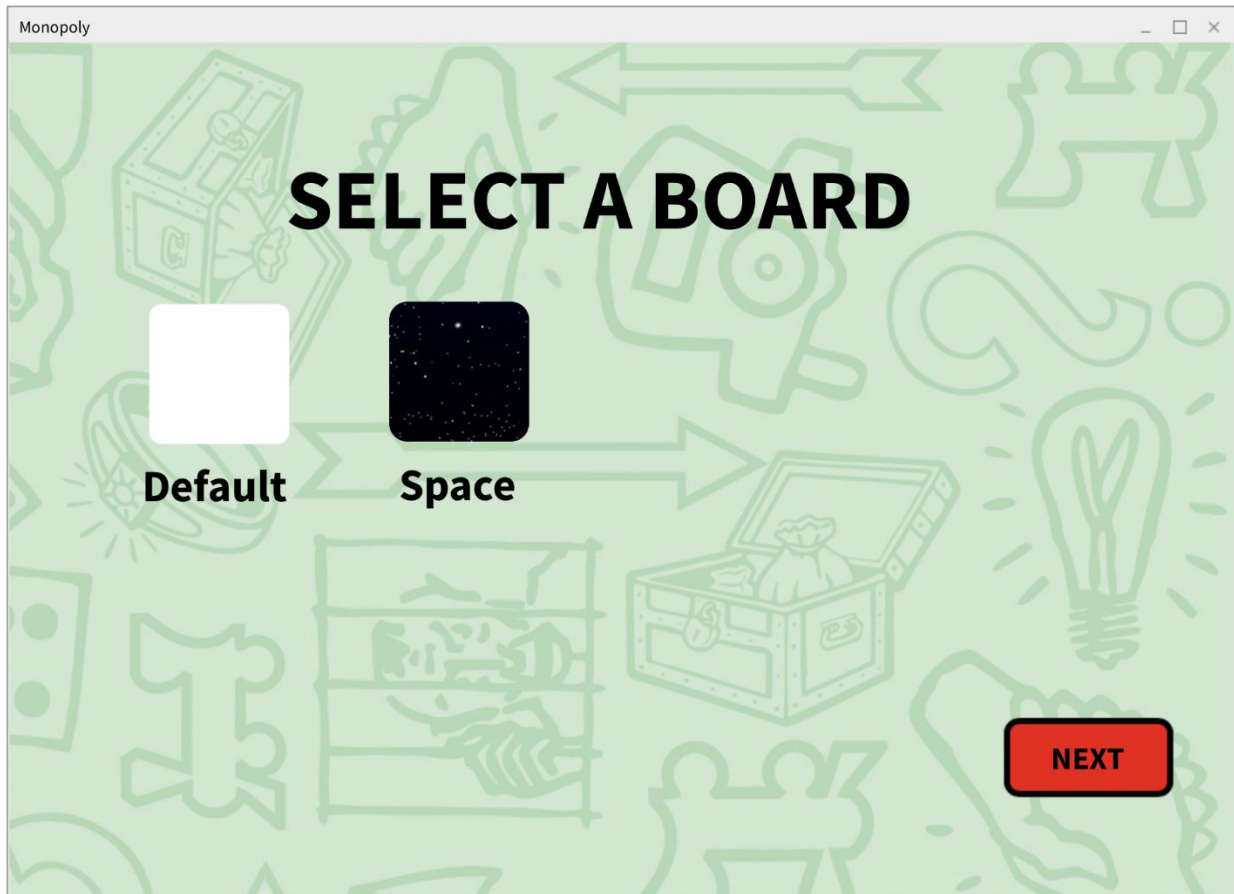


Figure 14. Board selection menu screen mockup

Board Selection Menu appears right after the user chooses whether they would like to play the game or edit a board. In both scenarios, Board Selection Menu comes and displays the saved boards. From this menu, the user can choose which board they would like to edit or play. During the first run, the Board Selection menu only has the default board. The users can edit the default board according to their preferences and save their changes as a new board or they can save it on top of the default board. In this mock-up screen, there are two boards, Default board and Space board. In this scenario, the user had already edited the default board and saved the changes as a new board, called "Space".

6.4.2.4 Settings Screen

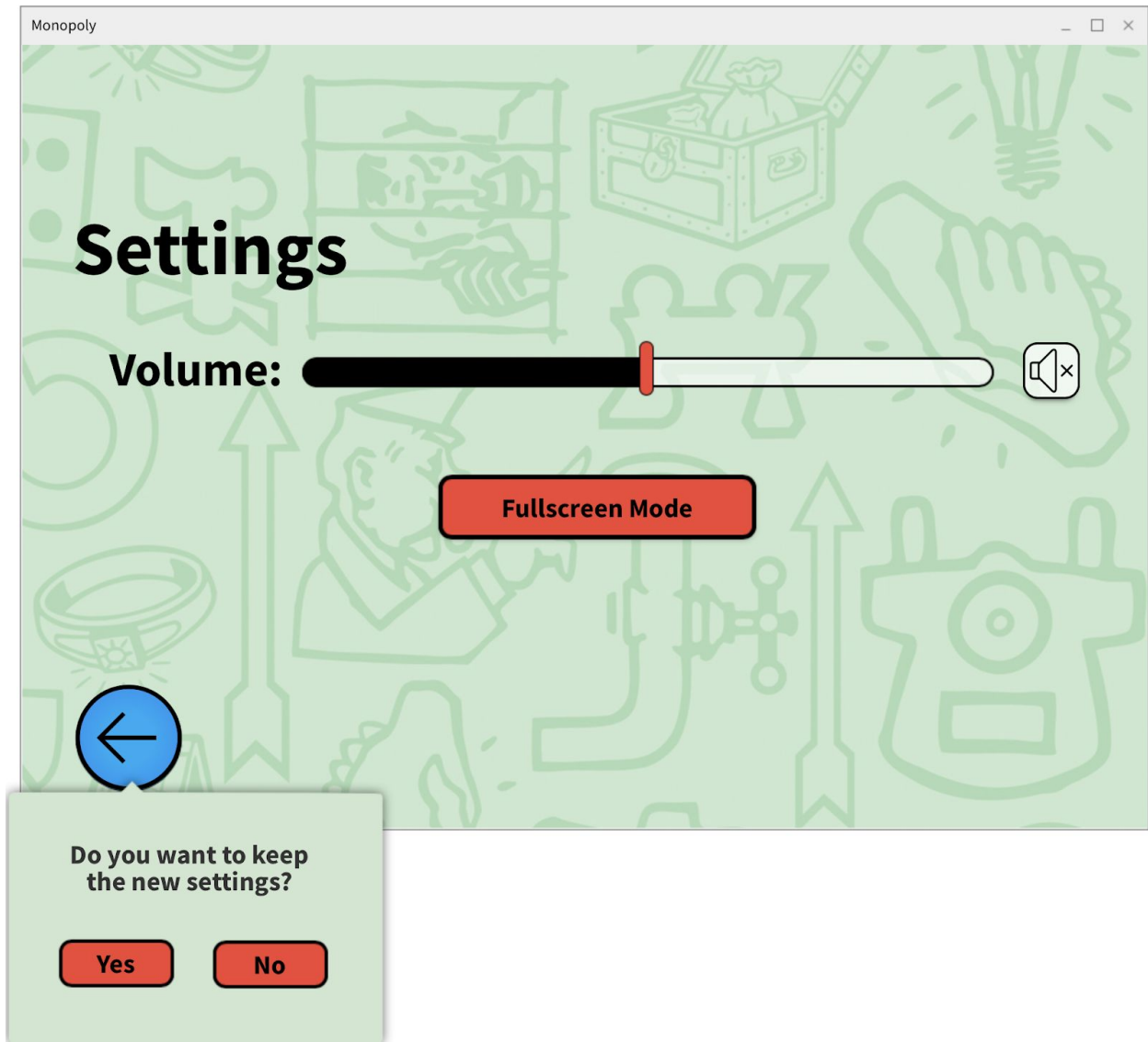


Figure 15. Settings screen mockup

This screen shows the settings of the game. Players can adjust the volume using the slider or they can enter the fullscreen mode using the “Fullscreen Mode” button. When pressed, this button turns into a “Exit Fullscreen Mode” button. Additionally, when the player presses the arrow at the bottom left of the screen, the player can exit the settings screen after he/she answers the confirmation window that pops up.

6.4.2.5 Credits Screen

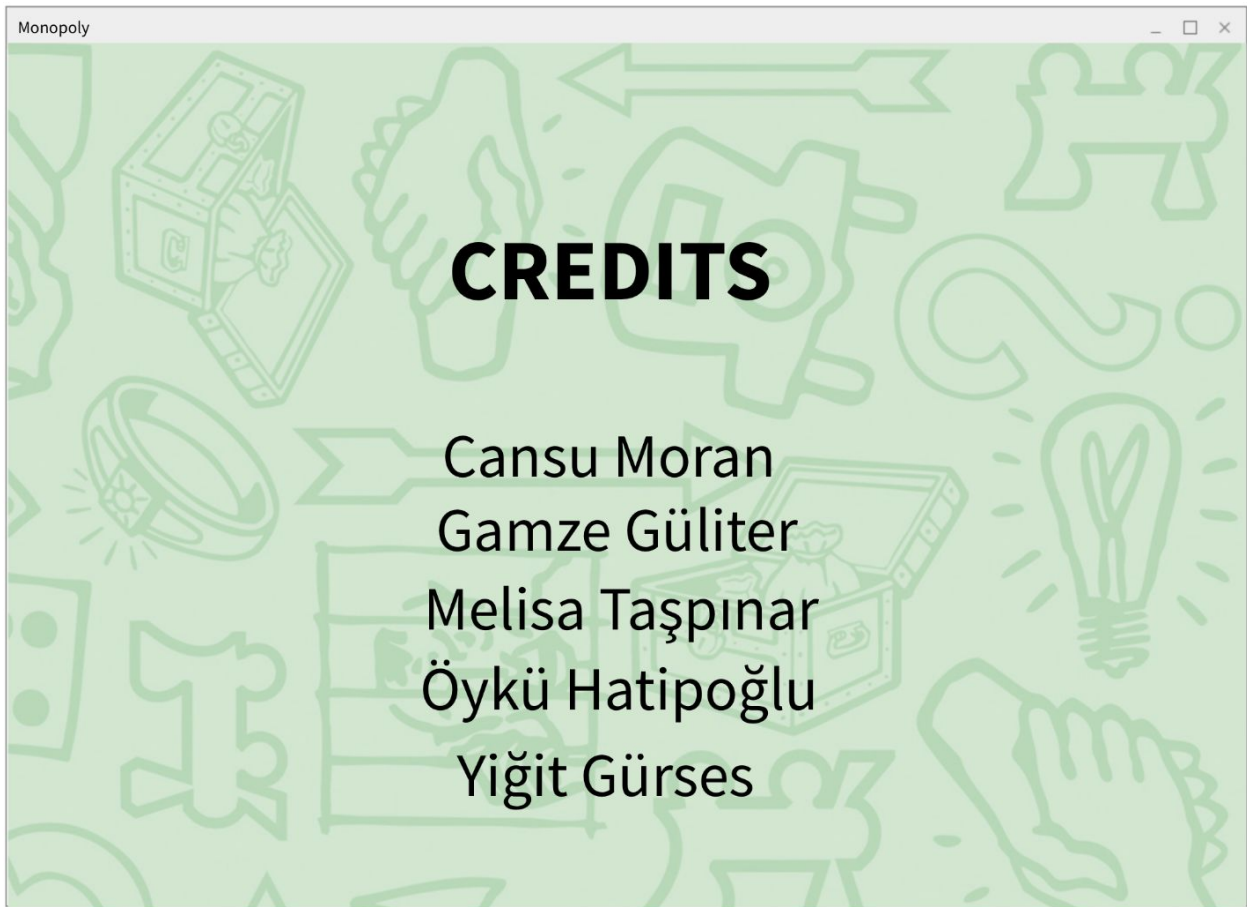


Figure 16. Credits screen mockup

Credits screen displays the names of the creators of the game.

6.4.2.6 Help Screen

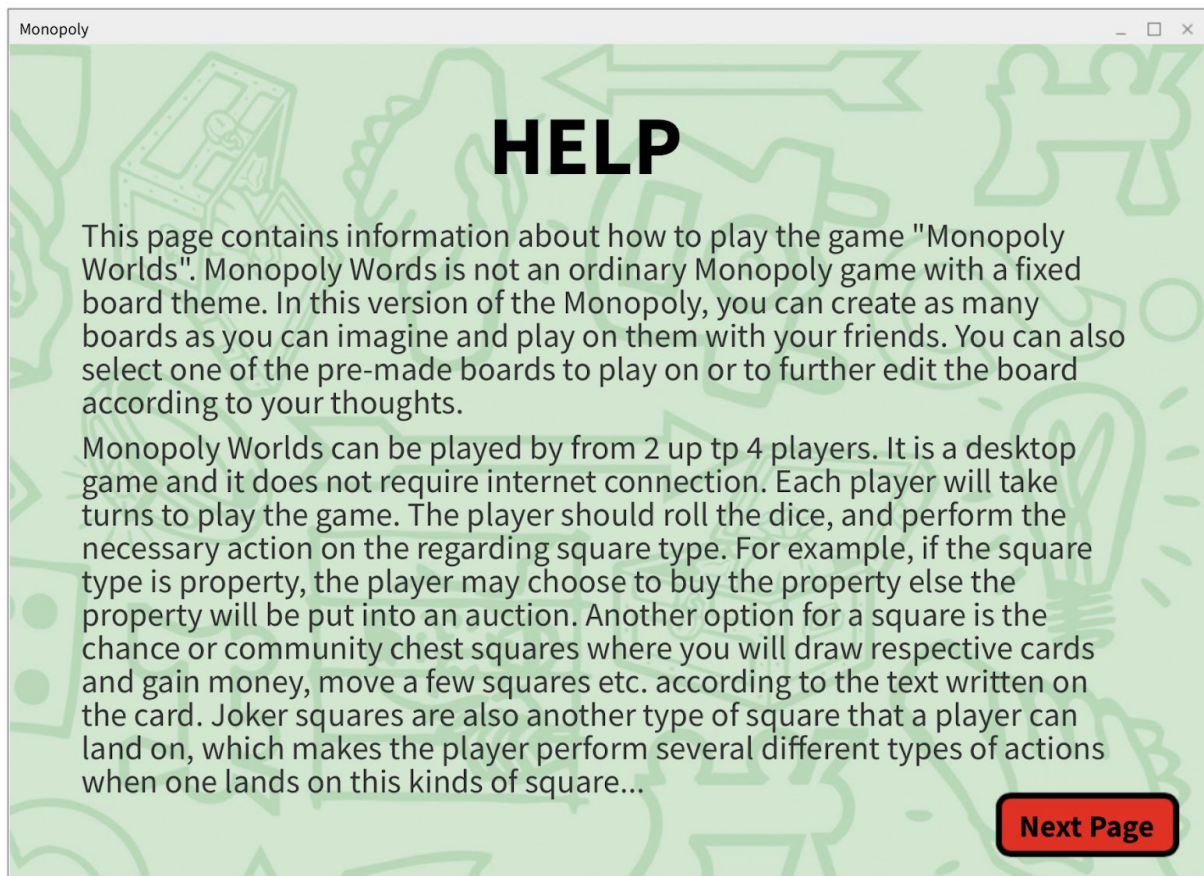


Figure 17. Help screen mockup

Help screen will contain an informative text about how to play the game and how to edit a board.

6.4.2.7 Edit Screen

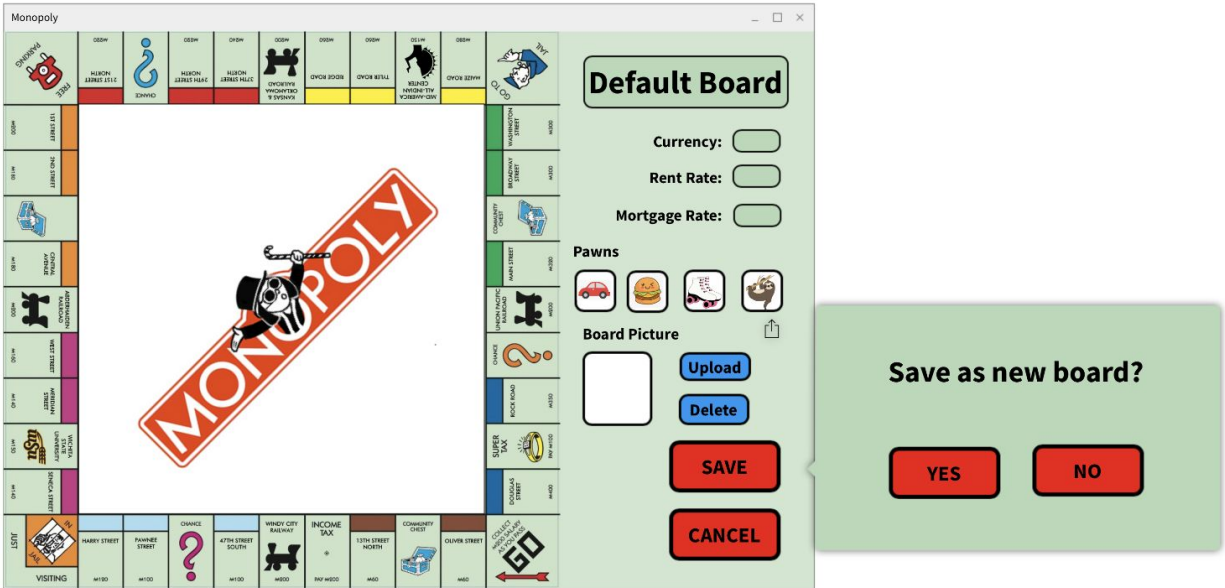



Figure 18. Edit screen mockup

Edit screen is where the user can edit the board they have selected from the Board Selection Menu. Information such as the name of the board, the currency name and rent and mortgage rate on the game can be edited directly from the Edit screen, as displayed above. The user can also add a board picture, which will be placed in the middle of the board behind the Monopoly logo, and will be used as the icon of the board on the Board Selection Menu. To edit the squares, the user should click on the square that they would like to edit and a pop-up screen will appear. After completing all the changes, the user can save the board as a new board, or they can save it on top of the board that they have selected. Additionally, the user can add up to 4 pawn pictures to be used on the game.

Select square type:

- ☒ Property
- ☐ Joker
- ☐ Chance
- ☐ Community Chest



When a user selects a square to edit, a pop-up screen will appear. This screen will ask the user to select a square type. A square can be one of four square types: Property square, Joker square, Chance square and Community square. After selecting the square type, the user can click on the arrow button to further edit the selected square, according to the square type.

Figure 19. Square type selection mockup

Property Square









Name:


Price:

Color Group: Blue Select

DONE


Color Groups

 Blue	 Brown	 Navy
 Green	 Yellow	 Red
 Orange	 Pink	

 **New Color Group**

New Color Group

Name:

Color: 

CANCEL DONE

Figure 20. Property square editing, color group selection, and color group addition mockups

If the selected square type is a Property square, the property edit pop-up screen will come. In this screen, the user can edit the name of the property as well as the selling price and color group of that property. If the user would like to change the color group of the property, they can press the “Select” button which will take them to another pop-up screen that will display all the color-groups currently existing on the game. The user can either choose from one of those pre-existing color groups or they can create a new color group by clicking on the plus icon, which will make the new color group generation screen appear. On this screen, the user can name their new color group and select a color for the group that will be displayed on the board.

Joker Square

Name:

Movement:

☐ Move squares

☐ Wait turns

☒ No movement

Money:

If the selected square is a Joker square, then Joker edit pop-up screen will appear. The user can edit the name and the picture of the square from this screen. In addition, they can edit the specific movement and money actions that this screen will have. For the movement, the user can pick whether they would like to make pawns that land on this square move a couple squares or make them wait for a couple turns. The user can't pick both options at the same time, however, they can choose not to pick either one of them as well. If they want to add a movement action, after deciding on which action to add, they can also change the number of squares or turns as well. The user can also add money actions to this square by typing the amount of money they want the players who land on this square to gain or lose. If the user doesn't want to add any money actions, they can leave the input box next to the money label empty.

Figure 21. Joker square editing mockup

Chance Square

Community Chest Square

Figure 22. Chance and community chest picture upload mockups

If the selected square is a Chance or Community Chest square, the user can only edit the picture of that square. The cards on the Chance and Community Chest decks will be automatically edited according to the currency entered on the Edit Screen.

6.4.2.8 Play Screen

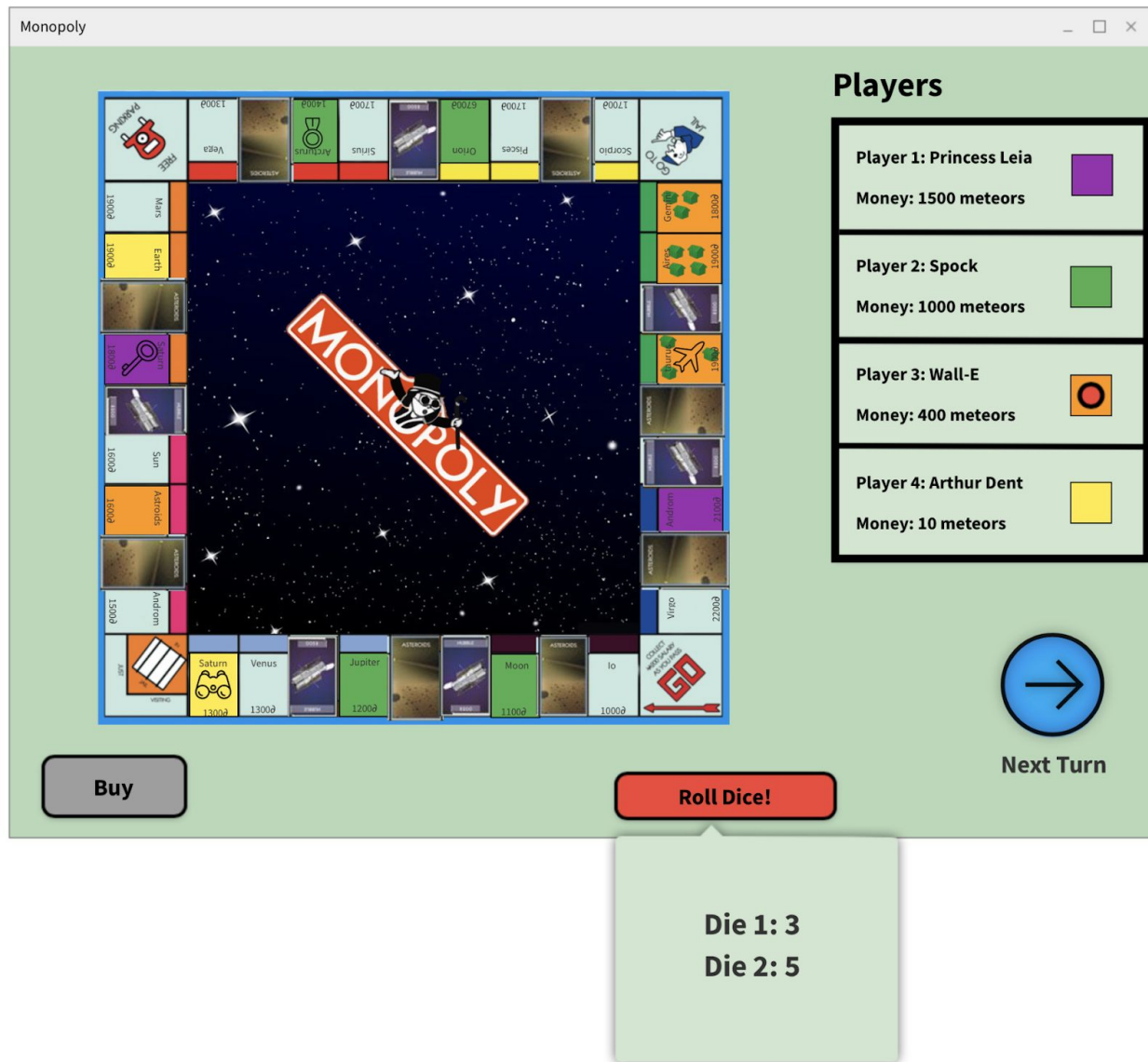


Figure 23. Play screen mockup

This screen shows a game in progress. On the board, there are some colored properties which represent the owners of those properties. For example, the yellow properties belong to the

player named “Arthur Dent” which can be seen from the player list on the right side of the screen. This list includes the names of the players, their money, and the color they use to mark their properties. Additionally, one can see whose turn it is by looking at the red circle on the colors of players. In this case, since the red circle is on the orange color, the turn is Wall-e’s. Looking at the board, we can see the green properties which are marked orange. They have plenty of houses on them and since it is Wall-E’s turn. Using the “Buy” button on the down-left of the screen, the player can buy the property he/she landed on if it’s not owned by another player. Since Wall-E owns the property and therefore cannot buy it, the buy button is inactive. To roll the dice, players can press the “Roll Dice!” button on the bottom of the screen., which will open a pop-up that has the results of the dice individually. When the player finishes his/her turn, the player will click on the “Next Turn” button.

6.4.2.9 Property Square Dialogs

Sirius	
Price:	1700
Rent:	1000
Rent with color set:	1200
Rent with one house:	1225
Rent with two houses:	1250
Rent with three houses:	1275
Rent with four houses:	1300
Rent with hotel:	1400
House price:	35
Hotel price:	200
<input type="button" value="Buy"/> <input type="button" value="Auction"/>	

Sirius	
Price:	1700
Rent:	1000
Rent with color set:	1200
Rent with one house:	1225
Rent with two houses:	1250
Rent with three houses:	1275
Rent with four houses:	1300
Rent with hotel:	1400
House price:	35
Hotel price:	200
House No:	0
Hotel No:	0
Owner:	Öykü
<input type="button" value="Sell"/> <input type="button" value="Add House"/>	
<input type="button" value="Cancel"/> <input type="button" value="Mortgage"/>	

Sirius	
Price:	1700
Rent:	1000
Rent with color set:	1200
Rent with one house:	1225
Rent with two houses:	1250
Rent with three houses:	1275
Rent with four houses:	1300
Rent with hotel:	1400
House price:	35
Hotel price:	200
House No:	0
Hotel No:	0
Owner:	Öykü
<input type="button" value="Pay Rent"/>	

Figure 24. Square dialog mockups for owned and unowned properties, for owned properties the view of properties from the owner's perspective and other players' perspectives

These dialogs show the property squares which pop up when a player lands on a property square. In the example on the left, the player landed on a property named "Sirius". On this dialog, the buying price of the property, the default rent and rents on different occasions are shown along with the prices of the houses and hotels. On the bottom of the square dialog there are two buttons that say "Buy" which makes the player buy the property and "Auction" which the player presses when he/she does not want to buy this property. This button will open an auction dialog. The other dialog in the middle shows the dialog of a owned square viewed by the owner of the square. It includes the same information with the unowned square dialog except on this dialog there is additionally, the number of the houses and hotels and the name of the owner is displayed. Also, on the bottom of the dialog there are the buttons "Sell", which makes the player sell this property, "Add House" button, which adds a house on the property square, "Cancel" button which closes the dialog, and "Mortgage" button which mortgages the property. Last

dialog on the right shows the view of an owned property square dialog viewed by a player that does not own this property. This dialog is similar to the dialog viewed by the owner of the property, however this time instead of other buttons, there is only a “Pay Rent” button.

6.4.2.10 Auction Dialog

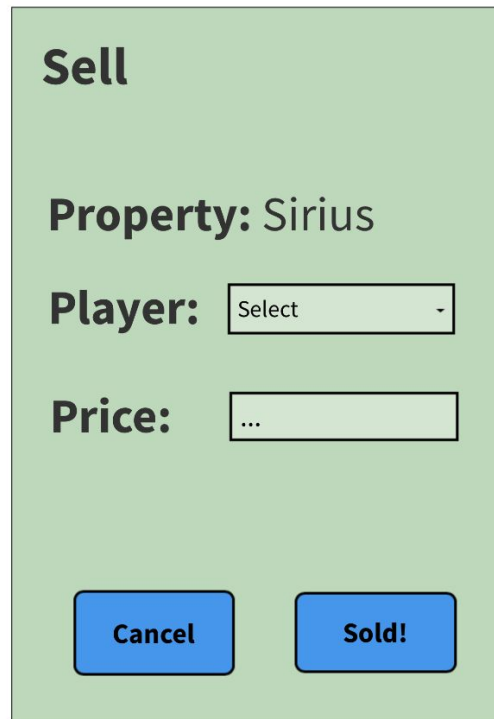


The image shows a mockup of an auction dialog box. It has a light green background and a black border. At the top, the word "Auction" is written in bold black text. Below it, "Property: Sirius" is written in bold black text. Underneath that, "Player:" is written in bold black text, followed by a dropdown menu with the word "Select" and a downward arrow. Below the dropdown, "Price:" is written in bold black text, followed by a text input field containing three dots "...". At the bottom of the dialog, there are two blue buttons with black text: "Cancel" on the left and "Sold!" on the right.

Figure 25. The auction dialog mockup

The user reaches this dialog by pressing the auction button on the square property dialog which pops up from the property he/she landed on and does not want to buy. In addition to this, an auction can also be initiated by the bankruptcy or resignation of a player. The properties of such a player will be taken into an auction. Under the “Auction” title, the name of the property is written on the dialog. After negotiating for the price and the player that will buy this property, users will fill this dialog with the name of the player that will buy the property and the price he/she will pay. Pressing the “Cancel” button the auction dialog will be closed if no one wants to buy this property.. Pressing the “Sold!” button the specified player will own the property paying the specified price.

6.4.2.11 Sell Dialog



The image shows a mockup of a 'Sell' dialog box. It has a light green background. At the top, the word 'Sell' is written in bold black text. Below it, 'Property: Sirius' is displayed in bold black text. Underneath, 'Player:' is followed by a dropdown menu showing 'Select' with a small downward arrow. Below that, 'Price:' is followed by a text input field containing three dots '...'. At the bottom, there are two blue buttons with black text: 'Cancel' on the left and 'Sold!' on the right.

Figure 26. The sell dialog mockup

In this dialog will be present when the owner of a property decides to sell his/her property to another player. Under the “Sell” title there is the name of the property to be sold. The player that will buy this property will be selected and the price he/she will pay for this property will be specified. If the owner of the property decides not to sell this property, the player can press the “cancel” button. To sell the property to the specified player for the specified price the “Sold!” button is pressed.

6.4.2.12 Joker Square Dialog

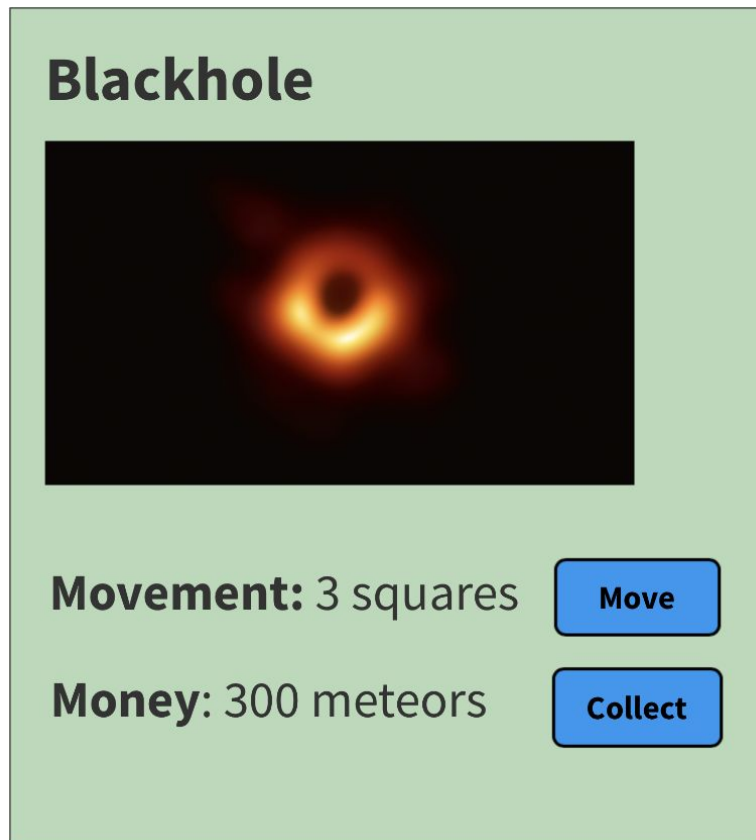


Figure 27. Square dialog pop up for a joker square named “Blackhole”

This dialog pops up when a player lands on a joker square. In this scenario, this joker square is the square titled “Blackhole”. The picture uploaded for this square is shown and under the picture there are actions of this square. This joker square has two actions, “3 squares” movement and “300 meteors” money. Since the player can choose to do these actions in any order that he/she wants, the buttons “Move” and “Collect” can be pressed any time to do the corresponding actions.

6.4.2.13 Chance Square Dialog

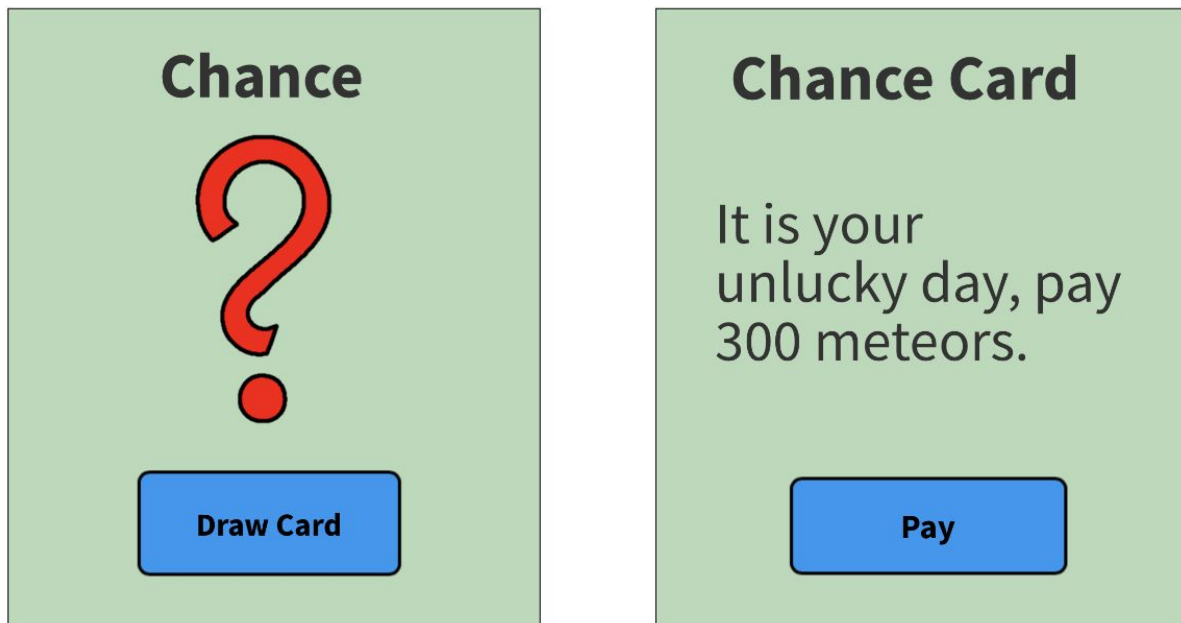


Figure 28. Square dialog pop up for chance square and dialog for chance card

This dialog pops up when a player lands on a Chance square. On the dialog on the left, the player can press the “Draw Card” button on the bottom to reveal the chance card. Then, a dialog that’s like the dialog on the right pops up to reveal the chance card drawn. This Chance Card has its own prompt and action. In this example, the card writes “It is your unlucky day, pay 300 meteors.” which shows that when the button on the bottom is pressed the player will pay 300 meteors. We can also see that the currency on the game is meteors.

6.4.2.14 In-game Menu

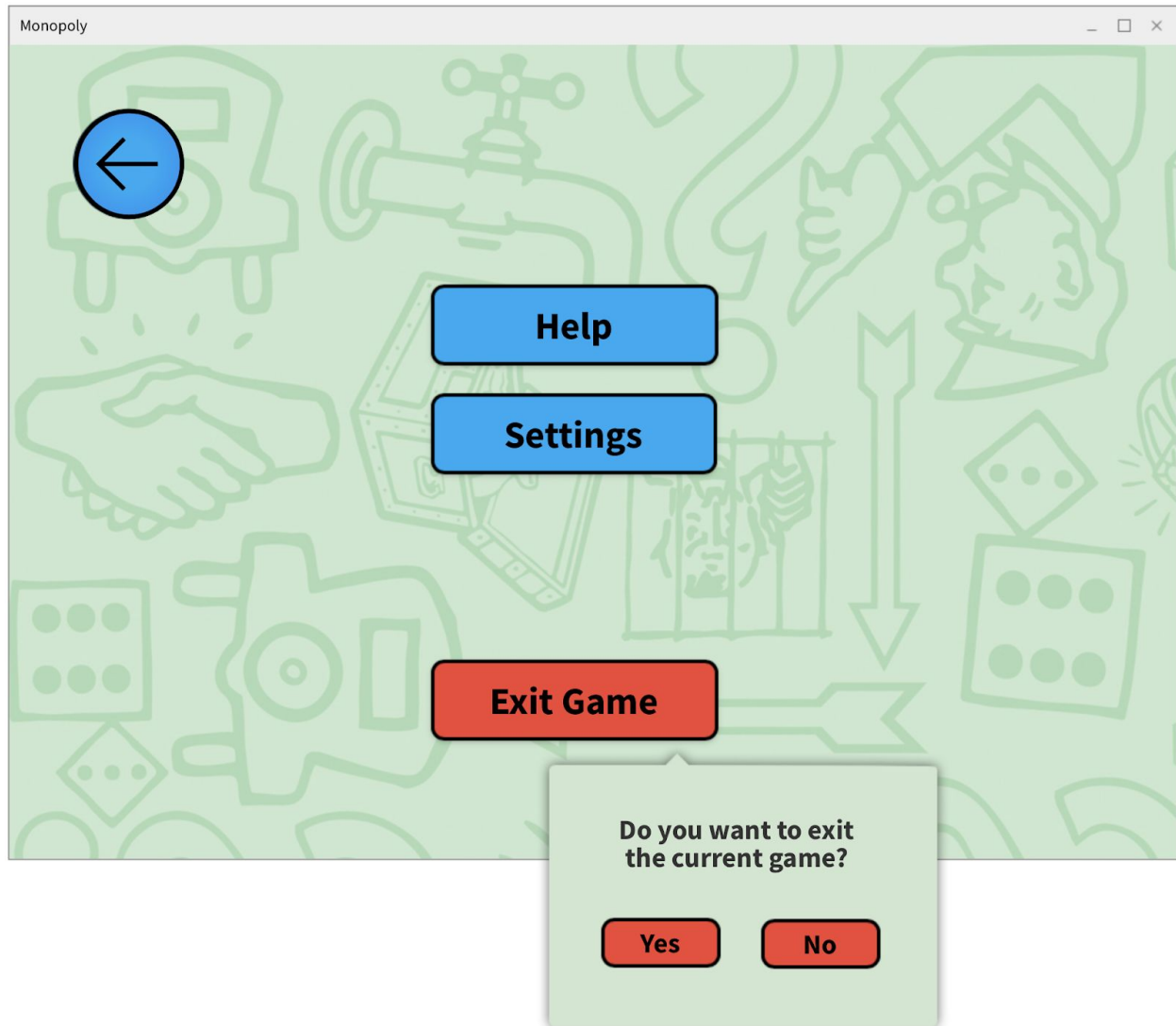


Figure 29. In-game menu mockup

This menu can be reached while playing the game or editing the board by pressing Esc button in keyboard. This screen contains “Help” and “Settings” buttons just like the main menu screen. The screen also has an “Exit game” button which will be “Exit Editing” when this screen is reached from the editing screen. This button will exit the current action (either play game or edit board) after displaying a confirmation window. Additionally, by pressing the arrow on the top left corner, the player can exit the in-game menu.

6.5 Priorities of Requirements

In our project, we decided to group the requirements according to their priorities. Implementing gameplay has higher priority than the editing feature since editing feature is only possible with a working game and without a playable game, no user would prefer our game. In addition, while implementing editing features, implementing editing features to Joker squares will have higher priority than implementing editing features to other squares. Joker places offer a more innovative perspective to the Monopoly game since players will be able to edit both game logic and user interface while editing Joker places. Following the Joker places, implementing editable features to Property squares will have the second highest priority. This includes, editable property square names, editable selling prices and editable color groups. At last, implementing editable features to Chance and Community Chest squares will have the least priority since there are not many editable features that Chance, and Community Chest squares offer.

7. Improvement summary

According to the feedback we got for our first iteration we updated the second iteration. We updated our sequence diagrams according to the changes we made in the implementation of the game and deleted the unnecessary ones such as sequence diagrams related to UI. We changed our screen mockups according to the changes in our implementation. For example, we removed or added some buttons for different screens and dialogs that we added. We added some additional non-functional requirements. We removed the UI related requirements that can change from user to user and not measurable such as time required to get used to the user interface. We changed functional requirements to better reflect the system requirements and revised the requirements according to the rule book. Any rule that was different on our requirements was revised, added or removed according to the rule book or the differences between the rule book and our implementation are written with reasonable explanation.

8 References

All the rules referred in our report were based on the official Monopoly rules in
"http://www.mtholyoke.edu/~blerner/cs315/Monopoly/MonopolyRules.pdf"