Bilkent University

Department of Computer Engineering

CS319 Term Project

Monopoly

Analysis Report

Project Group Members:

Melike Fatma Aydoğan

Ahmet Cemal Alıcıoğlu

Zübeyir Bodur

Beril Canbulan

Mehmet Çalışkan

Instructor: Eray Tüzün

Teaching Assistants: Barış Ardıç, Elgun Jabrayilzade, Emre Sülün

Table of Contents

Analysis Report

Monopoly Bilkent Edition

1. Introduction

In our CS319 OOSE Term Project, we chose to implement the Monopoly game. Moreover, we come with the design of the Monopoly with Bilkent University buildings so that we created the Monopoly Bilkent Edition. During the analysis process, while we thought about the technologies we should add, we tried to enhance the users playing experience.

2. Current System

Monopoly is a board game that players can enjoy to play together in their leisure time. In the game, players try to be rich as much as possible to bankrupt other players and win the game. Furthermore, the Monopoly game has many rules and versions, so that for our system, we choose some rules from the original game and add some rules to increase the fun.

3. Proposed System

3.1. Overview

3.1.1. Game

Our game allows friends to play Monopoly without physical meeting by being online. A friend will choose the Monopoly mode and start the game then will share the game code with other friends.

We improved the Monopoly experience with some improved features like trading and alliance.

3.1.2. Map

We designed our Monopoly with Bilkent University buildings. So that, we create another version of the Monopoly game.

3.1.3. Chat

To increase the communication of the players, the system has one or two chat sections. In the alliance mode, for team communication a private chatting box will be available. Besides that, in each mode a general chat box will be available for all players' communication.

3.2. Functional Requirements

3.2.1. Multiplayer and Online Game

Our Monopoly is a multiplayer game. Players can play this game online with their friends without physical interaction. So that they need the internet and computer during their play.

3.2.2. Opening a game

A player can open a game in three ways. New game can be started. With this option, the system will create a game code and this code will be used by other players. Another way is attending an already created game by using the game code. At last, the player can continue a game that he or she initialized. After that, they will share the code again with other users.

3.2.3. Choosing game mode

While the player starts a game, he or she needs to choose the game mode whether to be as with alliances or not and to be with speed dice or not. After the choice process game code will be shared on the screen.

3.2.4. To start the game

Each player should choose their pawn and if they are in alliance mode they, also, should enter their team number. Once the players have entered the necessary information, dice will be rolled to choose the order of players' turn.

3.2.5. Turns

In a turn, players can make moves according to which tile they are in and where they will land after the rolling dice. If they are in jail, they can pay the penalty and leave the jail with rolling dice or they can roll dice to get doubles for not paying a fine. A player can stay in jail upto 2 turns if they cannot get doubles in the third turn they need to pay fine. Then they will move according to their rolled dice.

- If they are not in the jail, they will roll dice and move according to it. The game board includes six types of tiles. Property tiles, tax piles, card tiles, jail tiles, GO pile, parking tile.
- If a player lands in one of the property tiles, the player can buy there if no one has owned this property. If the player does not want to buy there, the property will be sold with auction. If another player has already owned this property, rent should be paid to the owner.
- If a player lands in one of the tax tiles, the player needs to pay the tax to the bank.
- There are two types of cards, chance card and community chest card. According to the landed type of card, a card will be shown to the player and the player will play according to the card.
- There are two types of jail tiles. One of them is the jail tile and the other one is the go-to-jail tile. If a player lands this tile after rolling dice it is just visiting the jail and the player does not needs to wait any turn in here or pay fine. However, in go to jail tile, player needs to locate in jail and play the next turn according to the given rules above.
- If a player lands in GO tiles or pass from there, he or she will receive 2000 TL.
- If a player lands in parking tile, the player does not need to do anything.
- They can buy properties, pay taxes, buy houses and hotels, enter and exit jails and trading. After they did everything they need to finish their turn.
- Besides players locations, they can buy houses and hotels and do tradings.
- For buying houses and hotels, the player should have all properties of the same color to build into a property of that color.

3.2.6. Finishing or Saving the Game

Monopoly finishes when only one player has properties and money when other players are bankrupt. However, since the Monopoly game is a long game we will add save and finish early options. For both options, a player should offer them and other players will accept or decline it. According to the majority, the game will continue or end. For the finish early option, the system will sum up the money and the values of properties and buildings. Winner will be decided according to this summation. Besides that, a player can choose to leave the game in such a situation, the player's properties will go to the bank.

3.2.7. Trading

The system has a section for trading. From this section players can offer trading to other players in two ways. The player can offer to sell an item or buy an item. In both offers, if the other player accepts trading they will exchange items or the other player can offer new value for the properties. Also, it is important that a player only can offer during their own turn.

3.2.8. Speed Die

In this mode, players are not rolling 2 dice, but they are rolling 3 dice where one of them is a speed die. This speed die contains one 1, one 2, one 3, one BUS and two MR. MONOPOLY surfaces.

If the surface of Speed Die comes:

1, 2, 3 : Just sum all three dice and move forward according to summation.

BUS : With this die, player can choose one of two normal dice to move forward or move as normal. For example if dices are came as BUS, 4 and 6, player can move forward 4, 6 or 10 tile.

MR. MONOPOLY : Firstly, players moving forward according to normal dice and acting according to their tiles then they are moving to the next unowned property and the player can buy it or put it up for auction. If there are no left unowned property then the player will move to the next property and will pay rent if it is owned by other players.[1]

3.2.9. Alliance

With choosing alliancing mode, players can be teams at the start of the game. With alliancing mode, team members will not pay rents to other team members.

3.2.10. Chat

Players can text in chat boxes to message other players. In the alliance mode only team members can text each other otherwise all messages will be public to all players.

3.2.11. Losing the Game

A player can continue to watch the game after losing the game. They will not be able to play their turn only, the system will skip their turn since they bankrupted.

3.3. Non-functional Requirements

Usability : The Monopoly game includes many various actions during the many different cases. Therefore, to decrease the confusion, on the screen we will put possible actions so that a player can decide what to do during his/her turn.

Reliability : Our Monopoly needs the Internet since it is played online. So that we will check each player's connections. Otherwise, system will give output related to it and pause the game. If player cannot connect to game again other players can vote and can continue to game. Otherwise, other players will be bored. Also, as stated in the usability, since the system shows the options during a turn, it will decrease the possibility of false input and increase the reliability.

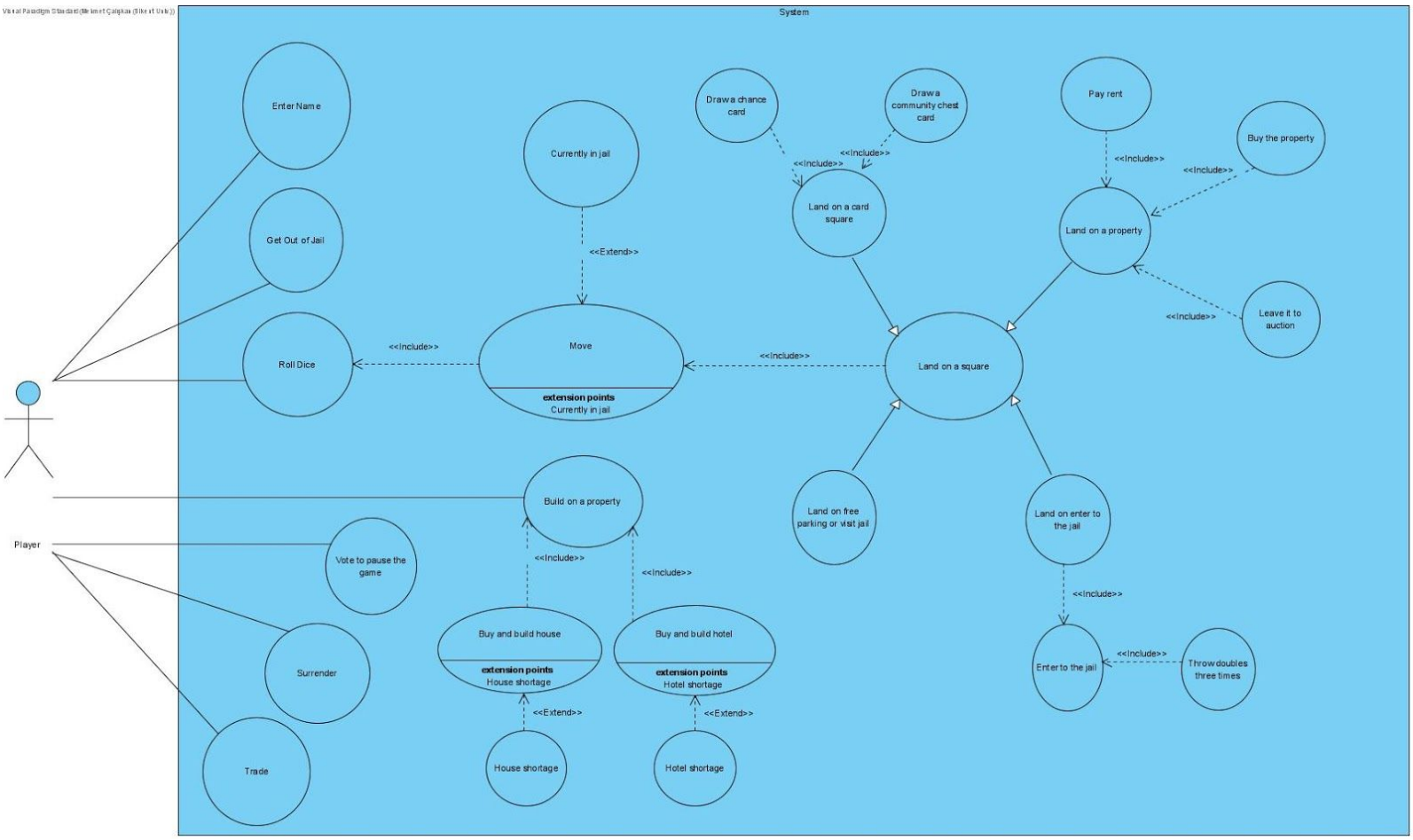
Performance : Our system will use both cloud storage and computer storage to get data. Cloud storage will be used for general ongoing of games which means it will be used mostly. Moreover, cloud storage can consume so much time, so that we will divide the information of game status to get the data as much as fast.

Supportability : Since our system is well designed according to the hierarchy and decomposition, necessary changes or additions can change the properties roles. Other than that, to maintain our system, we will store less data as much as possible during the save operation. Otherwise, our system can crash because of overflow.

Implementation Requirements : Our project was only restricted to Object Oriented Programming Languages and we chose to code in Java language and use its Swing library to implement the GUI.

3.4. System Models

3.4.1. Use-Case Model



Use Case #1: Create Game

Primary Actor: Player

Stakeholders and Interests:

- Player wants to create a game
- System creates the game session and places the player

Pre-conditions:

- Player must have entered a username.
- Player must have a proper internet connection.

Entry-conditions:

- Player clicks at the Create New Game button.

Event flow:

- Player clicks at the Create New Game button.
- Player chooses a game mode (Alliance/Speed Die).
- Player gives the Game ID to other players for them to join the session.
- After the game session has a minimum 2, maximum 6 players, Player clicks at the “Start the Game” button.
- System starts the game session.

Use Case #2: Join Game

Primary Actor: Player

Stakeholders and Interests:

- Player wants to join a pre-existing game session.
- System places the player to the specified game session.

Pre-conditions:

- Player must have entered a username.
- Player must have a proper internet connection.

Entry-conditions:

- Player clicks at the “Join Game” button.

Event flow:

- Player clicks at the “Join Game” button.
- Player enters the Game ID to the text field.
- If the Game ID is correct, System places the player into the pre-existing game session.

Use Case #3: Turn of a player

Primary Actor: Player

Stakeholders and Interests:

- Player wants to complete a turn.

- System processes the player actions.

Pre-conditions:

- Player must have the control.
- Player must not be in the jail.

Event Flow:

- Player clicks at the “Roll Dice” button.
- System processes the request and displays the result of dice.
- System moves the player to the specified square.
- Player completes the actions according to the square that he/she landed.
- System processes the actions that player have taken.
- Player clicks at the “End Turn” button or time limit reaches.
- System ends the turn and takes over the control to another player. (double?)

Alternative Event Flows:

- If player throws double:
    a. If this is the first or second double, System gives the turn to the player again.
    b. If this is the third double, System puts the player into the jail.

Use Case #4: Land on a not owned property square

Primary Actor: Player

Stakeholders and Interests:

- Player lands on a not owned property square.
- System processes the player actions.

Pre-conditions:

- Player must land on a property square.

Event Flow:

- Player lands on a not owned property square.
- System shows a choice panel which has 2 options: Buy or leave it to auction.
- If player clicks at “Buy” button and player has enough money to buy the property, System takes the specified money amount and assigns the property to the player.
- If player clicks at “Leave it to auction” button, System starts an auction process. After ending the auction, System takes the specified money amount and assigns the property to the winning player.

Use Case #5: Land on an owned property square

Primary Actor: Player

Stakeholders and Interests:

- Player lands on an owned property square.
- System processes the player actions.

Pre-conditions:

- Player must land on an owned property square.

Event Flow:

- Player lands on an owned property square.
- If the player has enough money to pay the rent, System transfers the money to the property owner.
- If player doesn’t have enough money to pay the rent, System starts an trade process between the player and the property holder.
- After ending the trade process, System processes the requests.

Alternative Event Flows:

- If the player doesn’t agree with the property owner in the trade process, System bankrupts the player.

Use Case #6: Build Hotel or House on the property

Primary Actor: Player

Stakeholders and Interests:

- Player wants to build hotel or house to the property.
- System assigns the buildings into the property.

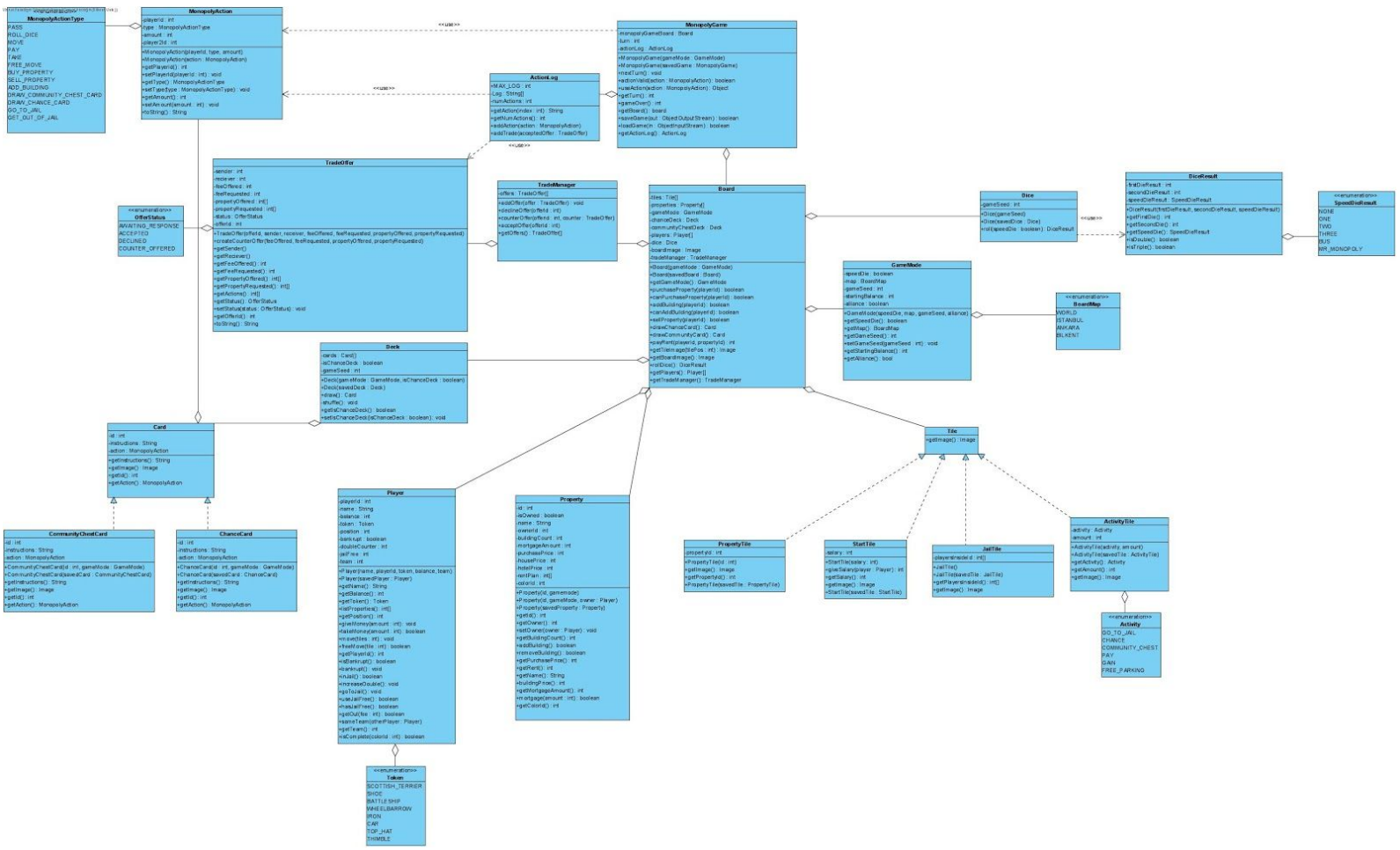
Pre-conditions:

- Player must own all properties in the color group.

Event Flow:

- Player clicks at “Build on Property” button and specifies which building type does he/she wants to build.
- System compares the money of the player and cost of the building. System also checks if there is a building shortage or not.
- If these conditions satisfy, System assigns the specified building type to the property and takes the money from the player.

3.4.2. Object and Class Model



3.4.3. Dynamic Models

3.4.3.1. Sequence Diagrams

3.4.3.1.1. Scenario: Player's Turn

Below diagram shows the outcomes when a player rolls their dice. The user presses the roll dice button from the GUI and rollDice() is called from the Board class. Then, depending on the *doubleRolls*, either the player is sent to Jail and sendJail() method is called, or the player's turn is being played.

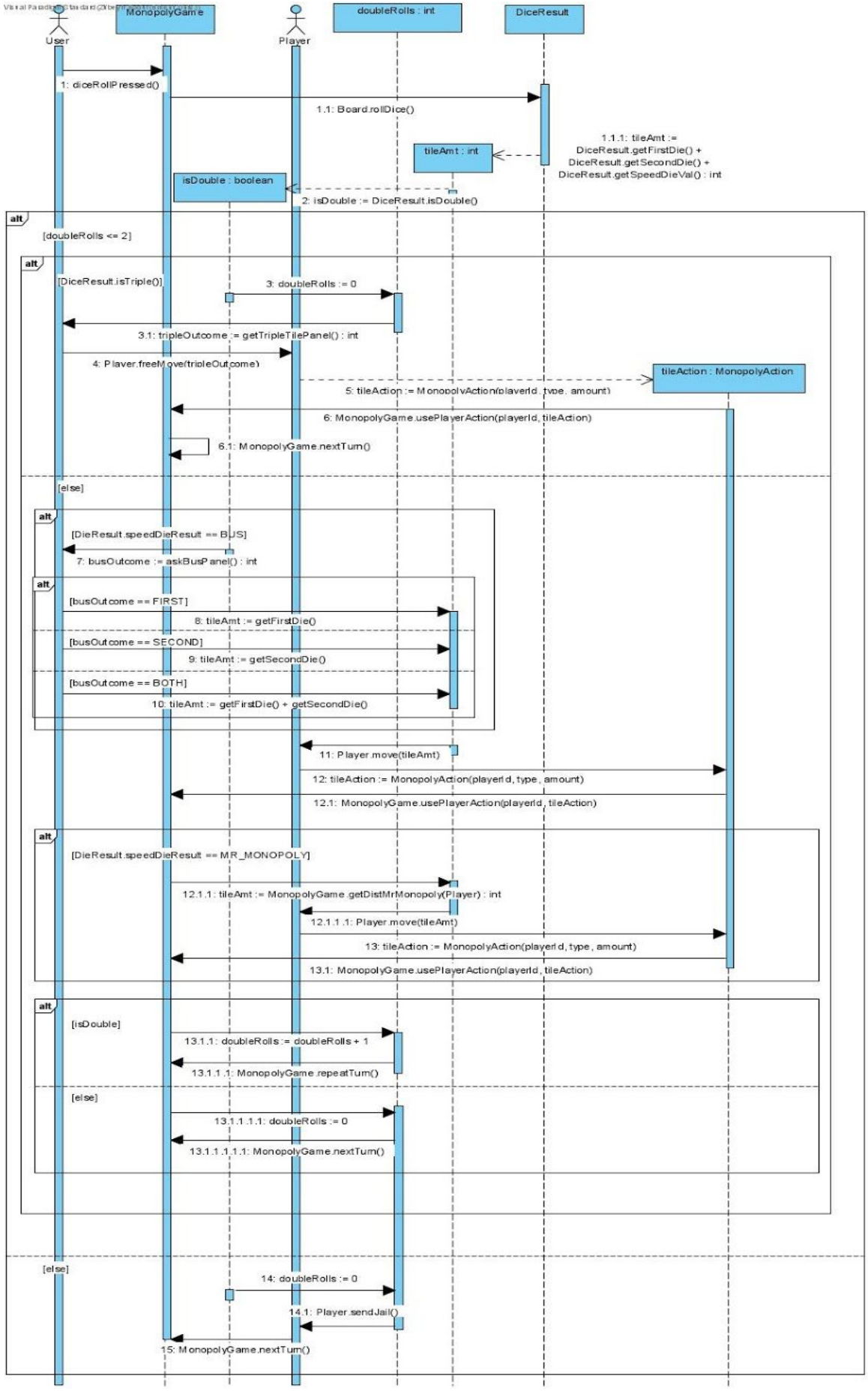
When the player hasn't made enough *doubleRolls* to be sent to Jail, their DiceResult is checked. First of all, whether the DiceResult.isTriple() checked. Then if the dice is a triple, the player chooses a tile from the board and makes a freeMove(). Else, the following algorithm is used to decide the *tileAmt*.

First, speedDieResult is checked again. If it's a BUS, it's asked from the User, who controls the Player, to choose either of the dice or both of them, using askBusPanel(). Then *tileAmt* is changed from sum of the first & second die and the speed die value, to either first or second die's value or sum of both of them.

Then, the player makes their move using move(tileAmt). After their move, the speedDieResult is checked again: if it's a MR_MONOPOLY, they need to move to the closest owned/unowned property. The *tileAmt* is computed using getDistMrMonopoly(Player). Then a move(TileAmt) is made again, if it was a MR_MONOPOLY.

Finally, after all moves and MonopolyActions, whether the DiceResult.isDouble() checked. If it is, *doubleRolls* is incremented and the user retains their turn using repeatTurn(). Else, *doubleRolls* is reset and nextTurn() is called to pass the turn to the next Player.

In addition, after each move, *tileAction* is chosen as MonopolyAction(playerId, type, amount) and the action is realized using usePlayerAction(playerId, tileAction) method.


User
MonopolyGame
Player
doubleRolls : int
DiceResult
1: diceRollPressed()
1.1: Board.rollDice()
1.1.1: tileAmt := DiceResult.getFirstDie() + DiceResult.getSecondDie() + DiceResult.getSpeedDieVal() : int
tileAmt : int
isDouble : boolean
2: isDouble := DiceResult.isDouble()
alt
[doubleRolls <= 2]
alt
[DiceResult.isTriple()]
3: doubleRolls := 0
3.1: tripleOutcome := getTripleTilePanel() : int
4: Player.freeMove(tripleOutcome)
tileAction : MonopolyAction
5: tileAction := MonopolyAction(playerId, type, amount)
6: MonopolyGame.usePlayerAction(playerId, tileAction)
6.1: MonopolyGame.nextTurn()
[else]
alt
[DieResult.speedDieResult == BUS]
7: busOutcome := askBusPanel() : int
alt
[busOutcome == FIRST]
8: tileAmt := getFirstDie()
[busOutcome == SECOND]
9: tileAmt := getSecondDie()
[busOutcome == BOTH]
10: tileAmt := getFirstDie() + getSecondDie()
11: Player.move(tileAmt)
12: tileAction := MonopolyAction(playerId, type, amount)
12.1: MonopolyGame.usePlayerAction(playerId, tileAction)
alt
[DieResult.speedDieResult == MR_MONOPOLY]
12.1.1: tileAmt := MonopolyGame.getDistMrMonopoly(Player) : int
12.1.1.1: Player.move(tileAmt)
13: tileAction := MonopolyAction(playerId, type, amount)
13.1: MonopolyGame.usePlayerAction(playerId, tileAction)
alt
[isDouble]
13.1.1: doubleRolls := doubleRolls + 1
13.1.1.1: MonopolyGame.repeatTurn()
[else]
13.1.1.1.1: doubleRolls := 0
13.1.1.1.1.1: MonopolyGame.nextTurn()
[else]
14: doubleRolls := 0
14.1: Player.sendJail()
15: MonopolyGame.nextTurn()

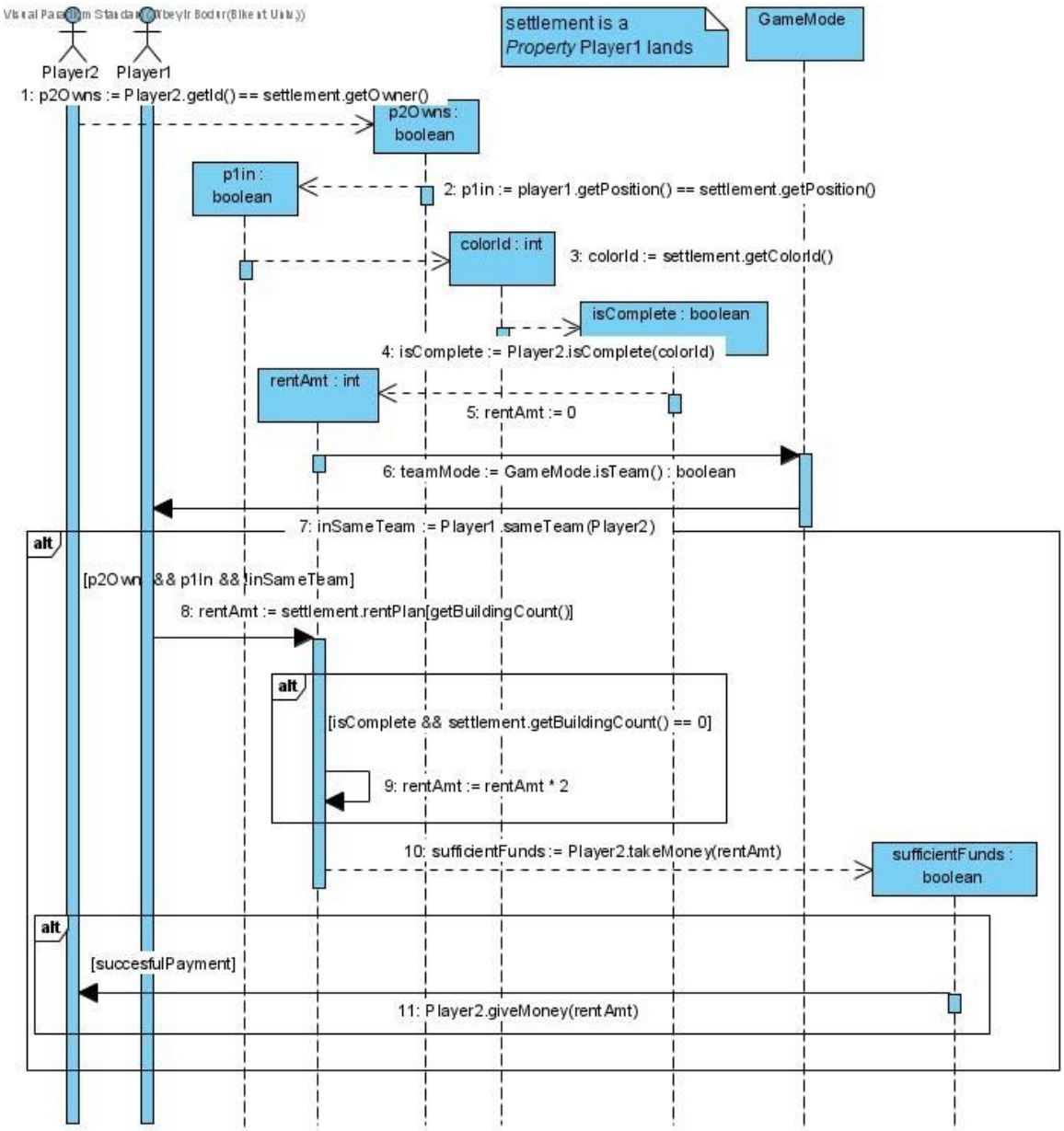
3.4.3.1.2. Scenario: Paying Rent

Below diagram shows how the amount of rent a player will pay is calculated when they land on another player's property. First, whether p2Owns this property, p1In (if Player1's token have landed this property) and Player1 and Player2 are inSameTeam() is checked. Then, if Player2 have collected the specific colour group for that *settlement* is checked (Player2.isComplete(colourId)).

After all this, the game is ready to compute the *rentAmt* for Player1. *rentAmt* is taken from the settlement.rentPlan(getBuildingCount()). So far, this will work for simple rent, houses & hotels. For the edge case, that if Player2 owns a colour group and has no house in the settlement, isComplete(colourId) helps: if it is true, *rentAmt* is doubled.

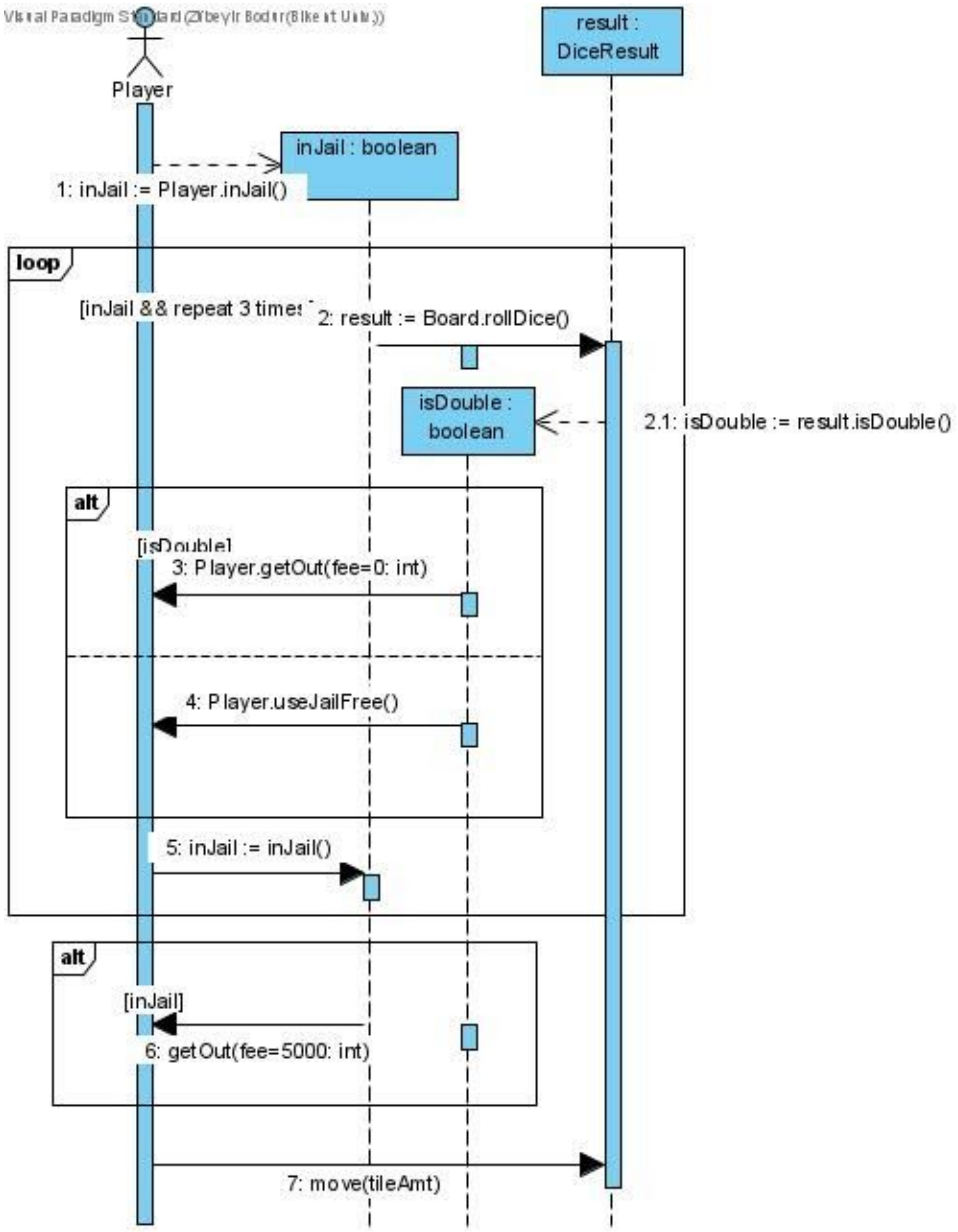
After computing the *rentAmt*, Player1.takeMoney(rentAmt) is called, if there is *sufficientFunds*, Player1 will be able to pay the rent and Player2.giveMoney(rentAmt) will be called.

When the *sufficientFunds* flag is false, scope will quit this diagram and the player will be forced to mortgage/sell their properties until it is true.


Player2
Player1
settlement is a
Property Player1 lands
GameMode
1: p2Owns := Player2.getId() == settlement.getOwner()
p2Owns :
boolean
p1in :
boolean
2: p1in := player1.getPosition() == settlement.getPosition()
colorId : int
3: colorId := settlement.getColorId()
isComplete : boolean
4: isComplete := Player2.isComplete(colorId)
rentAmt : int
5: rentAmt := 0
6: teamMode := GameMode.isTeam() : boolean
7: inSameTeam := Player1.sameTeam(Player2)
alt
[p2Owns && p1In && !inSameTeam]
8: rentAmt := settlement.rentPlan[getBuildingCount()]
alt
[isComplete && settlement.getBuildingCount() == 0]
9: rentAmt := rentAmt * 2
10: sufficientFunds := Player2.takeMoney(rentAmt)
sufficientFunds :
boolean
alt
[succesfulPayment]
11: Player2.giveMoney(rentAmt)

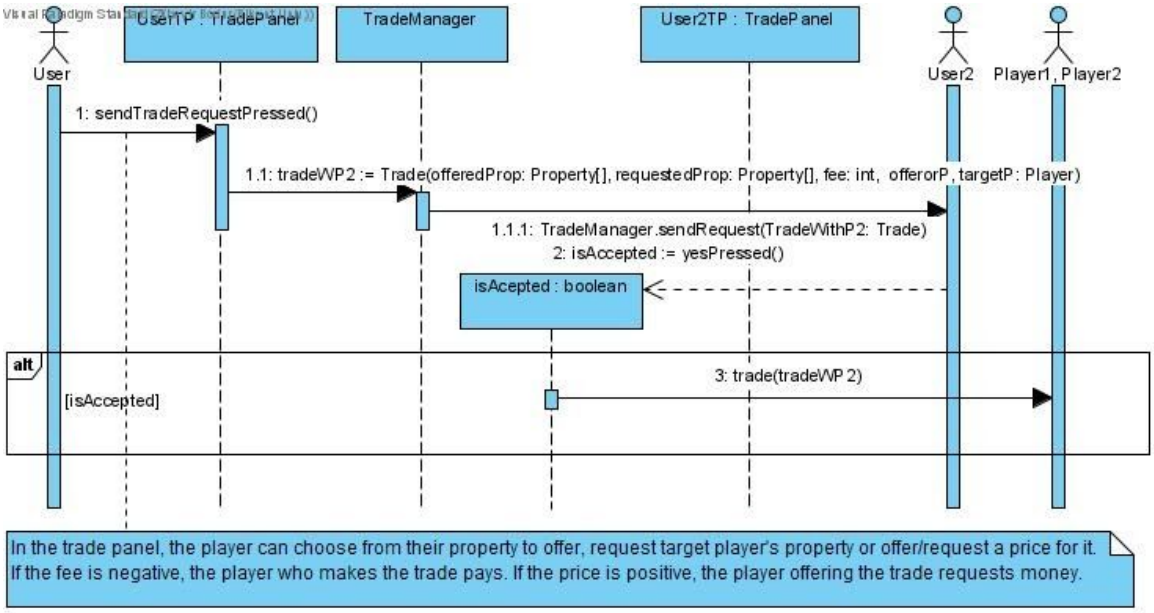
3.4.3.1.3. Scenario: Jail

Below diagram shows the process of how a player gets out of the Jail. If the player is inJail(), they rollDice() and if the outcome isDouble(), they getOut(fee := 0) of the Jail for free. If they were unsuccessful, they are also given the option to useJailFree() card to getOut(fee := 0). If the player fails to roll a double three times and can't get out of jail for free, they are forced to getOut(fee := 5000).



3.4.3.1.4. Scenario: Trading

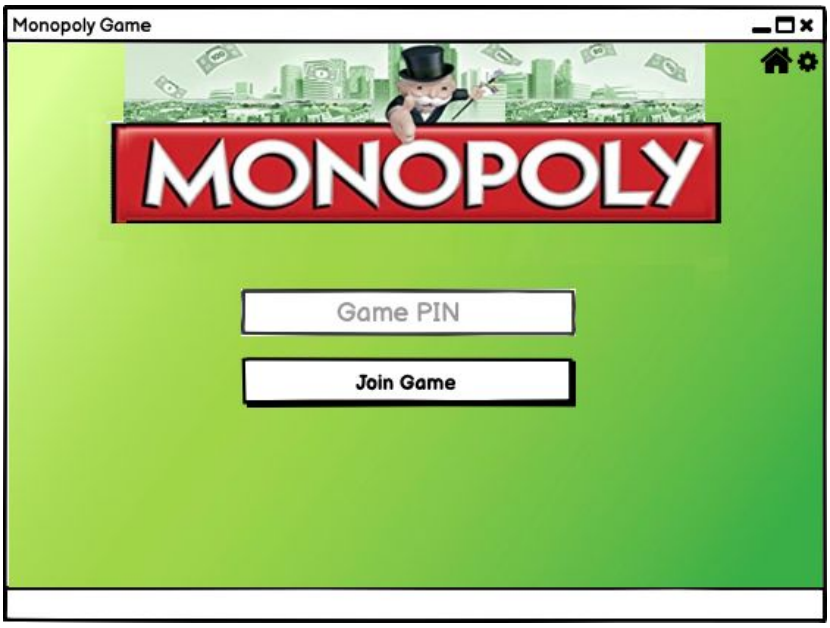
Below diagram shows the trade mechanism in the game. First the offerer User (user controlling Player1) chooses given values and properties to trade in the UI. Then, Player1 sends the request and sendTradeRequestPressed() is called, which initializes the Trade object. Then, TradeManager sends the trade request to the User2. If it *isAccepted* trade is realized by calling trade(tradeWP2).



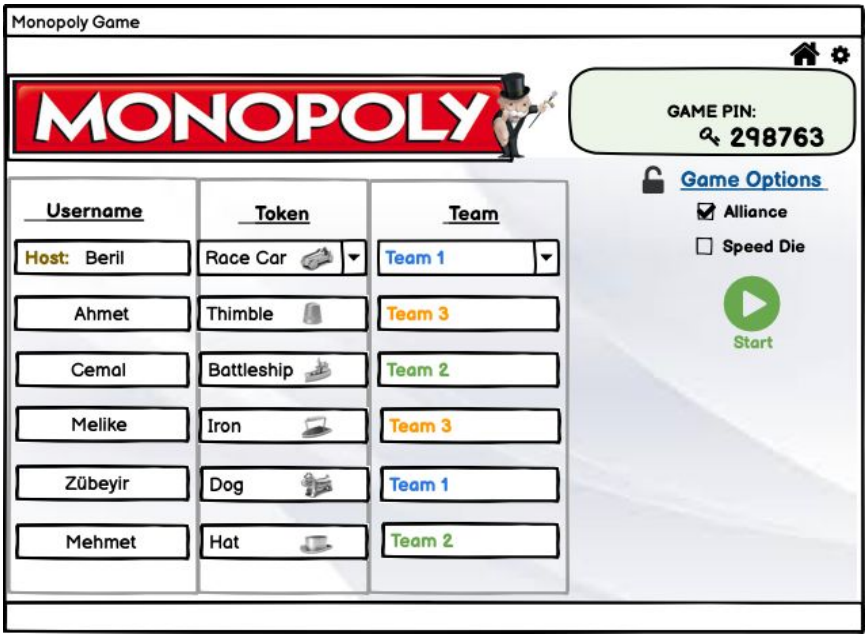
3.4.4. User Interface

Main Menu:

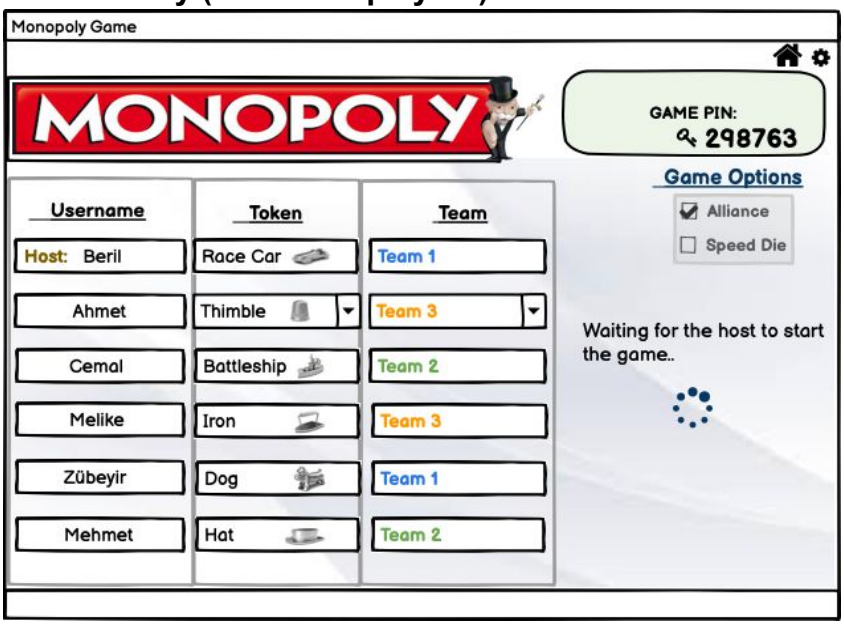
Join Game Screen:



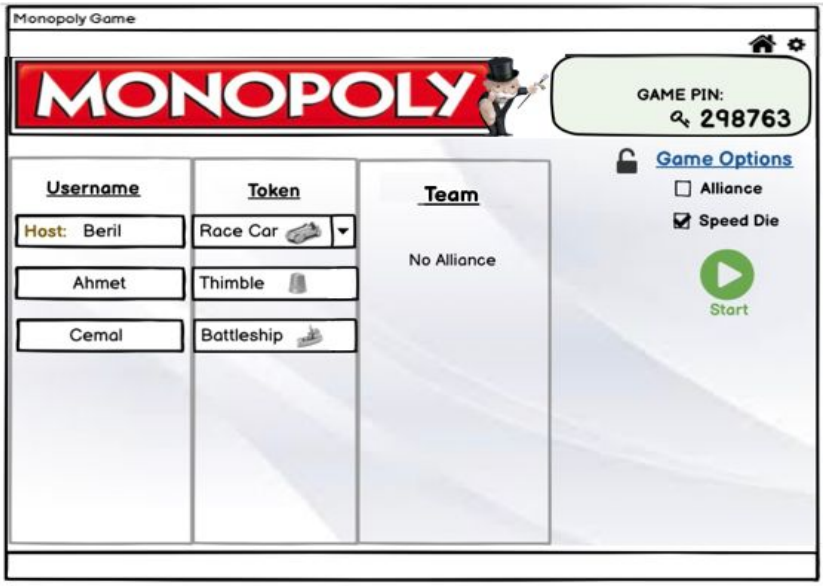
Game Lobby (For the Host):



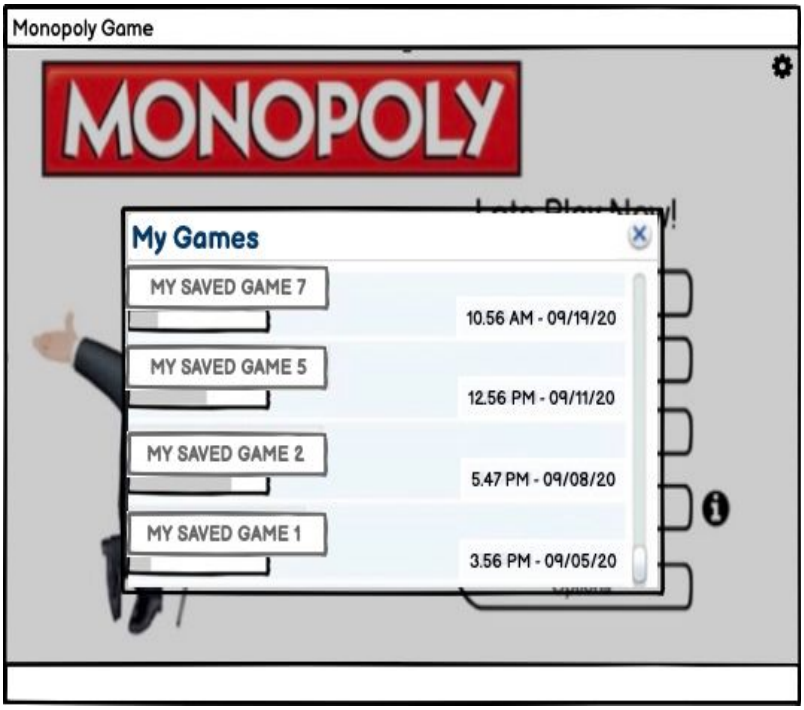
Game Lobby (For Other players):



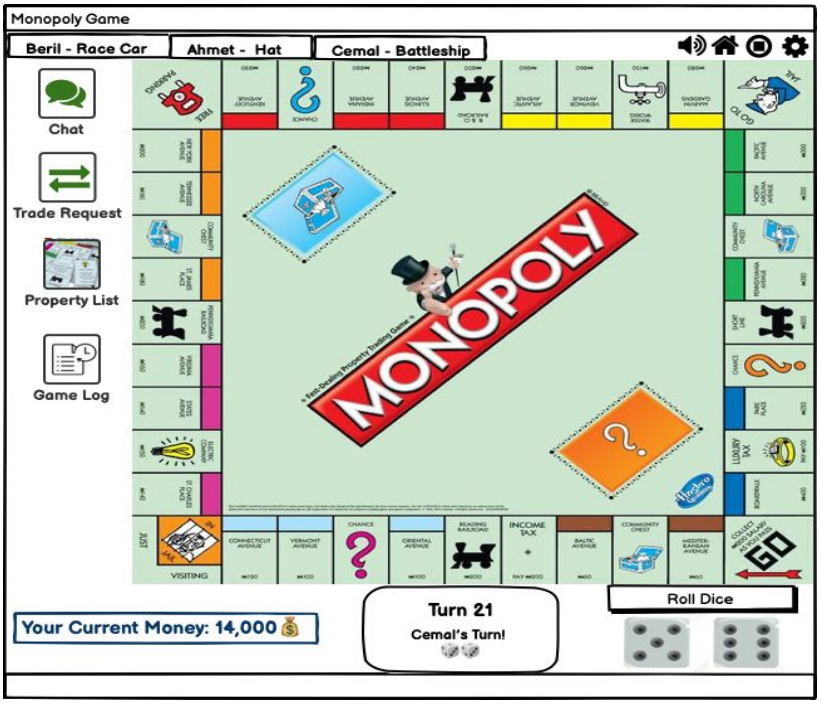
Game Lobby (Alliance Disabled):



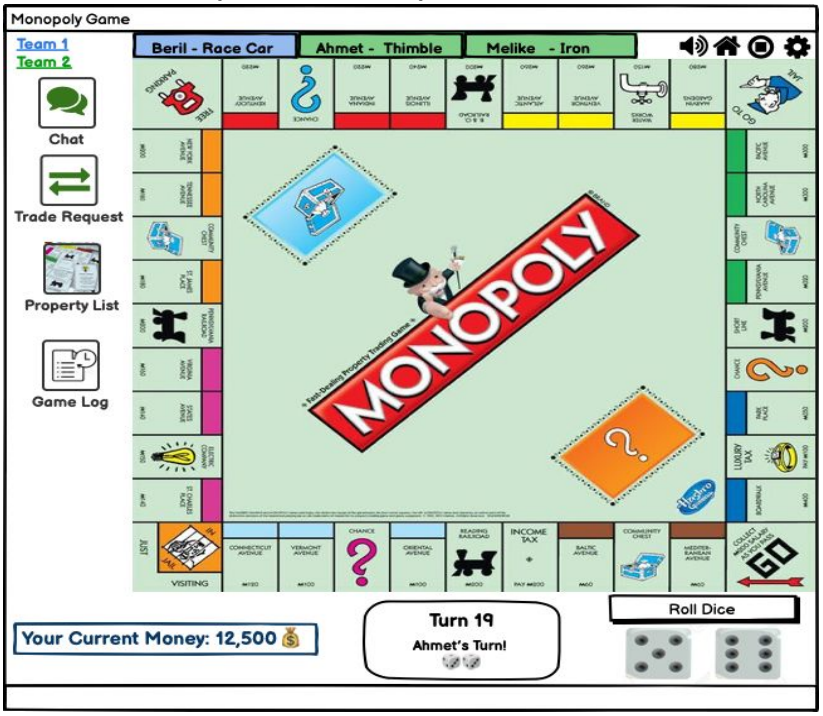
Load Saved Game:



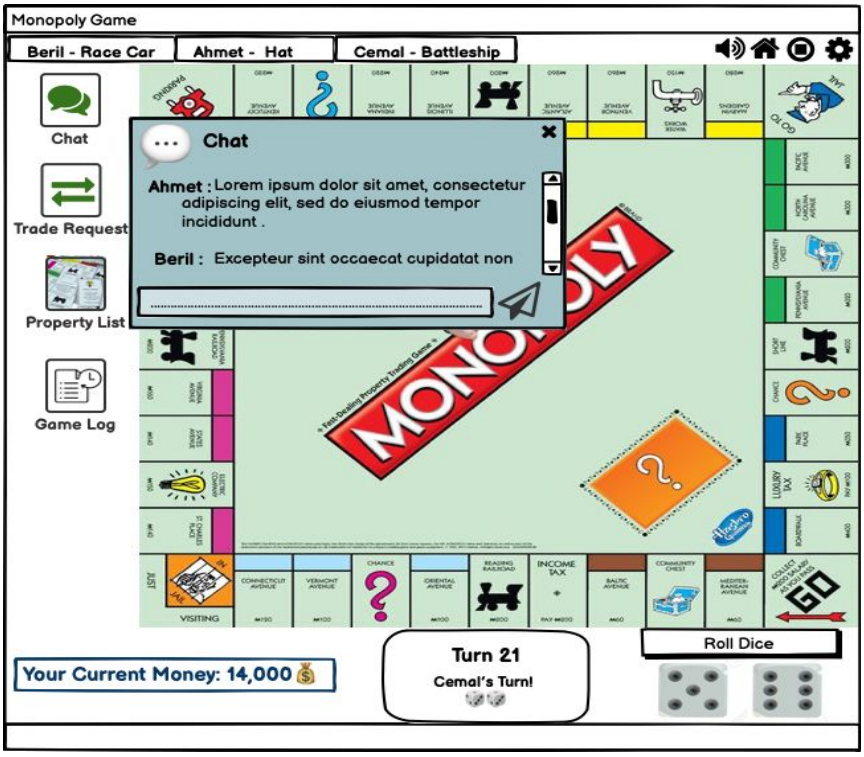
Game Screen:



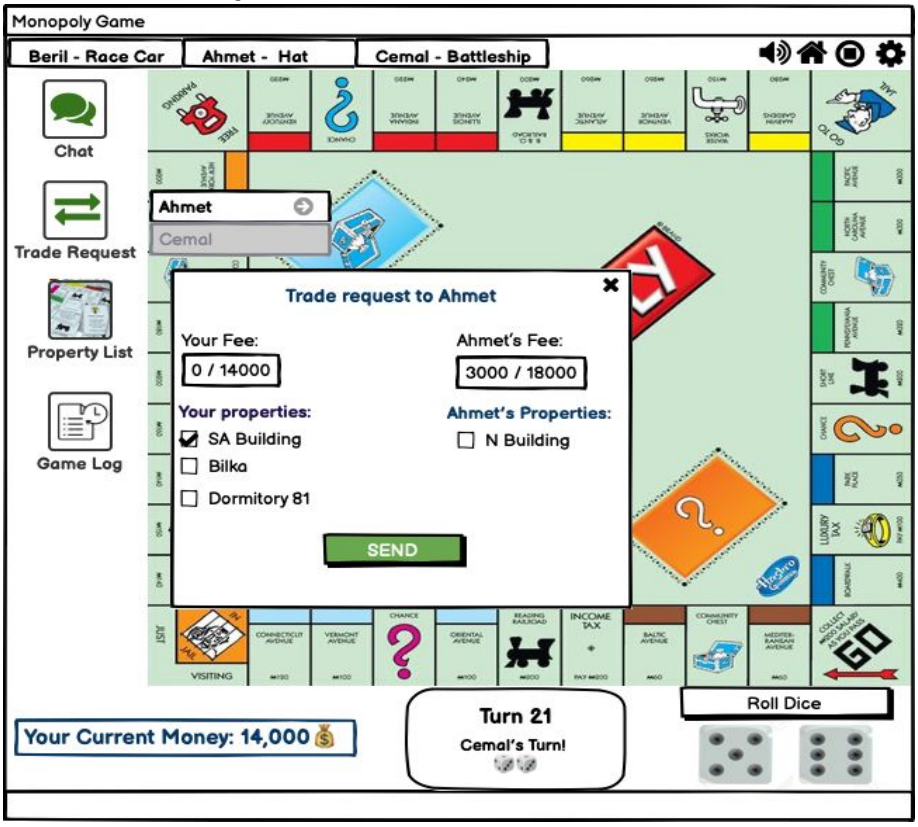
Game Screen (with Alliance):



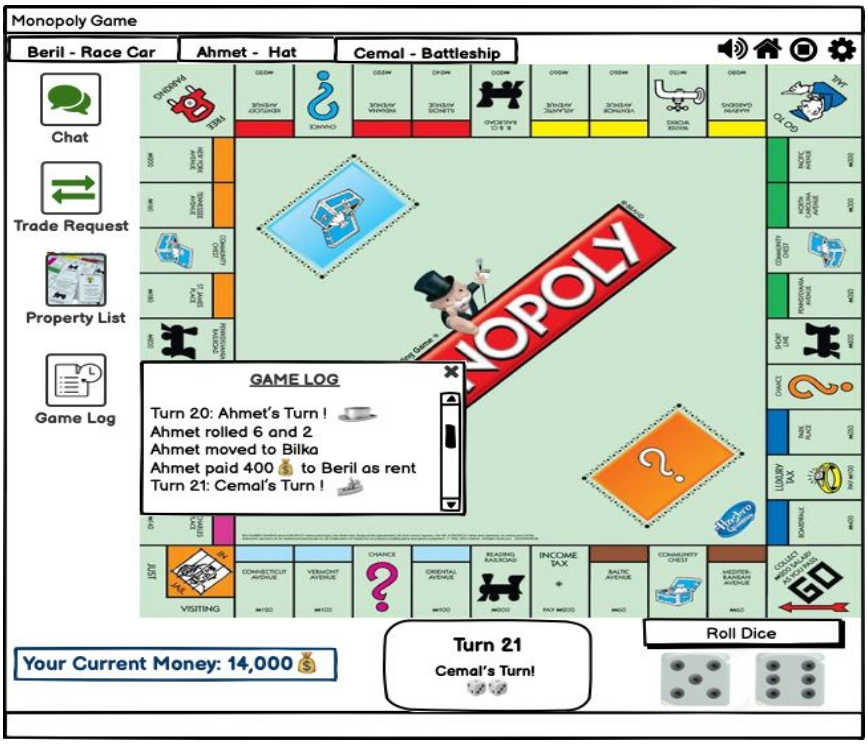
In Game Chat:



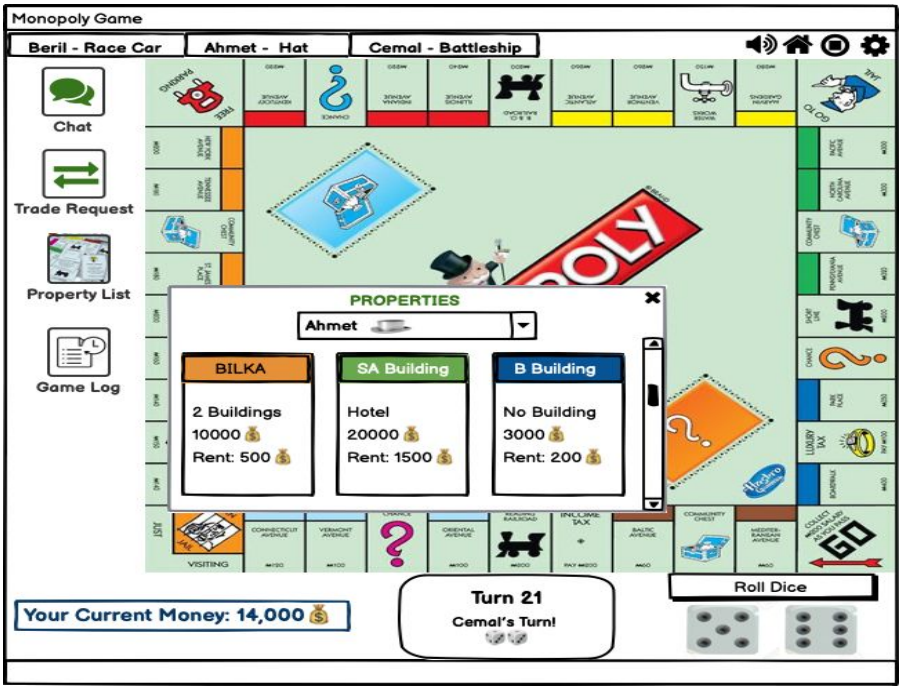
Send Trade Request:



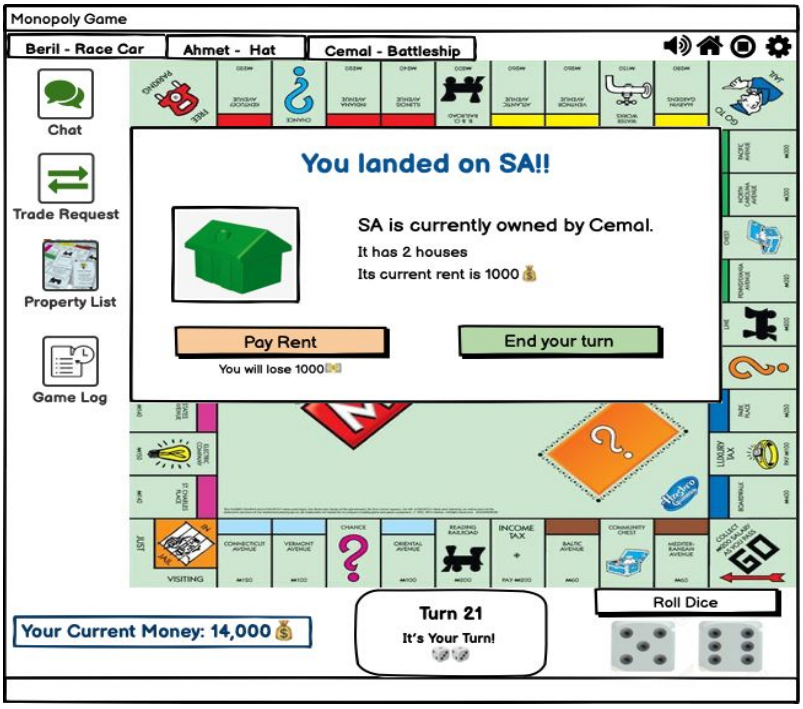
Game Log:

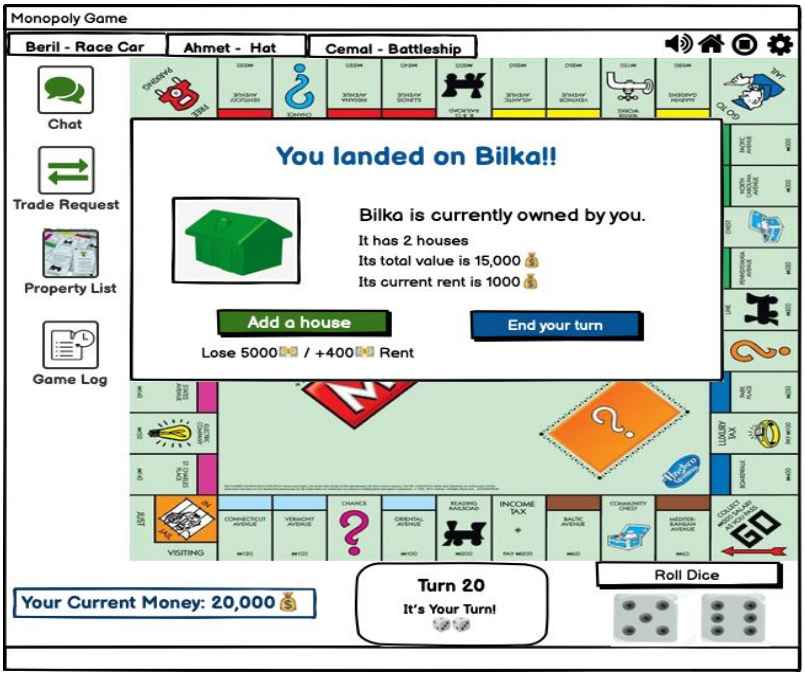


See Property List:



Game Choices:





4. References

[1] “Speed Die”. Accessed on : Nov. 1, 2020. [Online]. Available: https://boardgamegeek.com/image/456407/monopoly.

[2] “Rules of Monopoly”. Accessed on : Oct 20, 2020 [Online]. Available: https://www.hasbro.com/common/instruct/monins.pdf