Bilkent University
Department of Computer Engineering

CS 319 Term Project: Monopoly

Section 01

Group 1A

ANALYSIS REPORT

Atakan Dönmez
Elif Kurtay
Musa Ege Ünalan
Mustafa Göktan Güdükbay
Yusuf Ardahan Doğru

Instructor: Eray Tüzün
Teaching Assistant(s): Barış Ardıç, Emre Sülün, Elgun Jabrayilzade

Table of Contents

Table of Figures

1 Introduction

We are Group 1A and we will be recreating the household tabletop game Monopoly to be playable on the PC platform but with our brand-new twists. In this reimagined version of Monopoly, you will be able to play with up to 4 people locally (offline, played on a single computer) or against the computer. The game will closely follow the steps of the original one however the game's main issues regarding how long it takes to finish and its monotony will be addressed with new additions such as the thief who tries to catch players and steal their money, special perks all unique to the tokens, the ability to set a turn limit or to save your game and continue later on by loading and more. The players will also be able to play with custom maps that they created.

The game will be implemented in Java and will be designed and created using the principles of object-oriented programming as taught in CS 319 closely to the extent of our capabilities by utilizing techniques and principles taught in the lectures.

2 Overview

Digital Monopoly is a digital recreation of the classic board game Monopoly. You will be able to play locally with your friends, against digital players, or both at the same time on your computer. Each player takes turns rolling dice to advance on the board and perform actions like buying property, building houses, paying rent, trading, mortgaging, and more. The goal of a player is to become the richest person in the game while forcing her/his opponents to go bankrupt through managing your assets and mischievous trade deals to build your monopoly. The winner of the game will be decided upon when all but one player goes bankrupt or forfeits, or the turn limit is reached, then the wealthiest player is crowned as the winner.

2.1 Start Game

Users will be able to create new games or load their saves of previous games. When creating a new game users will add players that are going to be controlled by themselves. If the user-controlled player count is less than 4 the remaining spots will be covered by players created and controlled by the computer.

2.2 Gameplay

Players begin the game with M1500 (M is an abbreviation for money). Each player will roll the dice to determine their turn. The player that has the highest dice sum will have the first turn, the player that has the second highest dice sum will have the second turn, and so on. When a player starts the game they will choose a token and roll the dice. The player will move their token the number of spaces they have rolled. After the player completes their turn the next player will choose another token, roll the dice and move. The game will continue like this: each player will roll the dice when it is their turn and perform an action. If a player rolls double, they will move their token as usual, perform their turn actions, then instead of ending their turn the same player will roll the dice again and will play another turn. If they roll double three times consecutively, they will go to jail. The space they land on will determine their course of action. Each time when a player's token lands on or passes over the GO space the bank will pay them a salary of M200.

When a player stands on a property that is owned by another player, they will pay rent to the owner. If the property is mortgaged, they will not pay any rent. If the player that holds that property owns all the properties of the same color and that property doesn't have any buildings on it, then the visitor will pay double rent.

When a player stands on a Chance Card Space or a Community Chest Space they will draw the top card from the deck and will decide if they are going to open the card or postpone the card without opening it. If they decide to postpone they can open it in a future turn. Otherwise, they will follow the instructions on the card. After the actions on the card are performed, the card will be returned to the bottom of the deck. The "Get Out of Jail Free" card is held to be used later and can be sold/traded to another player.

When a player stands on a Tax Space, either "Income Tax" or "Luxury Tax", they will pay the amount specified on the space. The player can also choose to "Trade", "Build", "Mortgage" and "Redeem" at any time during their turn.

The player ends their turn once they are done with their actions and turn decisions. After all four players play their turn the lap count increases.

2.3 Tokens

Each player will choose a token at the beginning of the game. Each token has positive and negative perks.

Thimble: + Tax multiplier of 0.8
- Rent collect multiplier of 0.8

Wheelbarrow: + Building cost multiplier of 0.5
- Salary change of -M50 (Total salary is M150)

Boot: + Property cost multiplier 0.8
- Salary change of -M100 (Total salary is M100)

Horse: + Rent collect multiplier of 1.3
- Property cost multiplier of 1.1

Racecar: + Bonus salary is M200 (Total salary is M400)
- Jail Time of 4 rounds

Iron: + Building cost multiplier of 0.8
- Rent pay multiplier of 1.2

Top hat: + Jail Time of 2 rounds
- Tax Multiplier of 1.2

Battleship: + Rent pay multiplier of 0.7
- Building cost multiplier is 1.5

2.4 Map

The map consists of spaces. In addition to the classic Monopoly map, there will be different maps available. Also, the players will be given a “Custom Map” choice where they can use their own maps that they have created.

2.4.1 Land Property Spaces

The land property spaces feature a tier system in which the higher a tier the property is its cost of buying and building increases but also the rent imposed on other players increases too. This is a general trend as tiers (along with property cost, rent, and so on) increase from the “GO” position to the other end all the way back to the “GO” position again in the clockwise direction. In the default map, the tiers from the lowest to the highest in colors are Brown, Light Blue, Pink, Orange, Red, Yellow, Green, and Dark Blue. These colors can be changed by the user in custom maps. When a player wants to buy a property that is not owned, they should pay the written price. After buying, they will receive the Title Deed Card. If they do not wish to buy the property, the bank will sell it at auction to the highest bidder.

2.4.2 Transport Property Spaces

The players that land on these spaces pay rent according to the amount of transport properties that the property owner owns.

2.4.3 Electric Company and Water Works Property Spaces (Utility Properties)

The rent a player is going to pay if they land on one of these Properties is determined by their dice roll sum. If the property owner owns only one of these properties, the player pays 4 times the result of the dice roll sum, or if the property owner owns both of these properties the player pays 10 times the result of the dice roll sum as rent in the default map but can be altered in user made custom maps.

2.4.4 Tax

The players that land on these spaces must pay the specified tax amount to the bank.

2.4.5 Chance and Community Card

The players that land on these spaces draw a card from the deck related to the space. These cards have commands that players are forced to follow. However, the player can choose open and use the card's effect on the current round or keep it to open on the following rounds. These cards can have effects both positive and negative like "Go to X space", "It's your birthday everyone pays you X amount", "Go to jail" and more.

Inside "Chance" cards, there will be a "Thief" card. This card will introduce a new character/player in the game. The thief will try to catch a player by being in the same space as another player. When the Thief catches a player, that player will get their money stolen. After stealing the Thief disappears from the game.

2.4.6 Jail

The players can be sent to jail if they draw a card that sends them there, land on the "Go to Jail" space, or roll double three times in a single turn. When a player goes to jail, they cannot collect their M200 GO bonus. There are three ways to get out of jail:

1. They roll doubles on any of their next three turns.
2. They use the "Get Out of Jail Free" card (they can buy it from another player if they do not have it).
3. They pay a fine of M50 before they roll the dice on either of his or her next two turns.

If a player does not get out of jail by their third turn, they must pay the M50 fine and get out. Actions like collecting rents, participating in auctions, mortgaging, redeeming and trading are not affected when a player is in jail.

2.4.7 Wheel of Fortune

If a player lands on a Wheel Of Fortune Space, they will spin a wheel and gain a random property, money, a house on one of their properties if possible, a “Get Out Of Jail Free” card or they may lose money or one of their buildings on one of their properties.

2.4.8 Go

All players start in this space at the beginning of the game. If the player lands on or passes this space they get paid an M200 salary every time they do. The player can collect the salary as many times as they pass in a single round. If they are sent to jail or used a card that specifically mentions that the player should not get paid then the player cannot collect the salary.

2.5 Build

When a player owns all properties of the same color, they may buy houses from the bank and put them in those properties. Players cannot put more than one house on any property until they have built a house on every property of the same color. This rule continues for the second, third, and fourth row. Players should first complete that row for a color to start another row. No more than four houses can be bought for each property. When a player has four houses on a property, they can return the houses they have and buy a hotel to put it on any property of the color-group. A property can only have one hotel. These actions can only be done while the player is standing on a property of the color of the property that the player wants to build on. For example, if the player lands on a red property it can build on all red properties if the player owns them all, but no other property tier can be built upon.

2.6 Trade

A player can sell any unimproved properties (except buildings) to any player at any price. If buildings are standing on any property of the same color, they must first sell the buildings to the bank to sell any property of that color. A player can get half the price they paid for the houses and hotels when they sell them to the bank. Houses on a property cannot be sold at once, they should be sold one by one, the reverse of the manner they were bought. All hotels on one color-group may be sold at once, or they may be sold one house at a time (one hotel = five houses), evenly, in reverse of how they were erected. The player can make offers to others and others can accept or decline this offer.

2.7 Mortgage

Unimproved properties can be mortgaged through the bank at any time. The mortgage value is printed on each “Title Deed” card. To lift the mortgage, the owner must pay the bank the amount of the mortgage plus 10% interest.

When all the properties of a color-group are no longer mortgaged, the owner may begin to buy back houses at full price. The owner may sell this mortgaged property to another player at any agreed price. If you are the new owner, you may lift the mortgage at once if you wish by paying off the mortgage plus 10% interest to the bank. If the mortgage is not lifted at once, you must pay the bank 10% interest when you buy the property and

if you lift the mortgage later you must pay the bank an additional 10% interest as well as the amount of the mortgage.

2.8 Bankruptcy

You are declared bankrupt if you owe more than you can pay either to another player or to the bank. If your debt is to another player, you must turn over to that player all that you have of value and leave the game. If you own houses or hotels, these will be returned to the bank and half their value will be paid to the creditor by the bank. If you have mortgaged property you also turn this property over to your creditor, but the new owner must at once pay the bank 10% of that property. The new owner may then pay the principal or hold the property until they lift the mortgage. If they hold the property in this way until a later turn, they must pay the interest again upon lifting the mortgage. You should owe the bank, instead of another player, more than you can pay (because of taxes or penalties) even by selling off buildings and mortgaging property, you must turn over all assets to the bank. In this case, the bank immediately sells by auction all property so taken, except buildings. A bankrupt player must leave the game.

2.9 Forfeit

A player can choose to forfeit the game at the end of their turn using the forfeit button. If a player forfeits the game, they will turn over all of their assets to the bank

2.10 Game End

The game will end when all but one player is bankrupt or the specified turn count is reached. The game can be restarted or saved to restart or see the scoreboard later. If the exit button is pressed the game can also be ended manually. A scoreboard showing all the players' net worth and their ranking in the game will be shown.

2.11 Settings

Music and game sound can be adjusted through settings.

3 Functional Requirements

3.1 Menu Functions

- The user will be able to adjust music and game sound
- The user will be able to view the names of the developers.
- The user will be able to create a new game.
- The user will be able to choose a pre-existing or their own custom map to play on.
- The user will be able to add human-controlled players to the game. The game will add computer-controlled players to compensate for the empty seats until a player count of 4 is reached.
- The user will be able to pick a lap limit
- The players will be able to pick their tokens. The users should not be able to choose any token that have been chosen by other players
- The user will be able to view the previously saved games by their number, date, length, and the name of the players and load one to play.

3.2 Gameplay Functions

- The player should roll dice to start their turn. The game will move the token according to the randomly generated dice result.
- The game will prompt the user to buy the space if the token lands on an unowned property. The user will be able to choose between buying the property or let the property be presented for auction.
- All players will be able to bid on auctioned property and the highest bid will win.
- The game will prompt the user to open the card if the token lands on Chance or Community Chest. The player will be able to choose if they want to open the card or save it to open at a later round. The game will force an action depending on the contents of the card.
- The player will be able to start a trade with another player during their turn. The player will be able to choose how much money and which properties they are offering and what they want in return. The player that receives the offer will be able to either accept the trade or refuse it.
- The game will prompt the user to spin the wheel if the token lands on Wheel of Fortune. The player will earn reward based on the result of the spin.
- The player will be able to build if they land on a space where they own all the properties of the same color. The player should only be able to build on the properties of that color. The player should not be able to build the second house on a property before they build one house each on the other properties of the same color. Similarly, the player should not be able to build the third house before building two houses each, the fourth house before building three houses each on the other properties
- The players will be able to mortgage their properties or redeem their mortgaged properties during their turn.
- The player will be able to view the assets of any player.
- The user will be able to restart the game.
- The user will be able to save the game.
- The user will be able to exit the game. This action will cause the game to end even if any of the end conditions are not met.
- The game will end when any of the lap-limit, 3-player bankruptcy or exit game conditions are met. The game will display the names, net worth, and ranking according to their net worth of the players when the game ends. The user will be able to quit or restart the game.

4 Non-functional Requirements

4.1 Usability

Any user who has previously played the original tabletop Monopoly game should be able to understand how the game plays out. The new features added into the game should be understandable by new users therefore such features should include explanations on them (such as the tokens including descriptions of their advantages and disadvantages during the token selection screen)

4.2 Reliability

As the game handles the saving and loading features locally, no errors that could possibly corrupt the local files should occur. The game should prevent the user from entering inputs that would change the behavior of the game (such as entering strings that would let the user escape out of the input operation and add lines of code).

4.3 Supportability

The implementation should not include any deprecated technologies or methods to make future maintenance and extensions easier. Commonalities in methods, properties and in general classes should be structured to include abstractions and interfaces to add extensibility to the project. There should be explanations for methods and large chunks of code to provide readability to future/other developers of the project so it is easier to read and change the code.

5 Pseudo Requirements

The implementation will be monitored by Git Version Control System. The implementations, changes, and reports will be pushed to our GitHub repository.

6 System Models

6.1 Use Case Model

Figure 1 provides the use case model for the monopoly game. We provide descriptions for different use cases below.

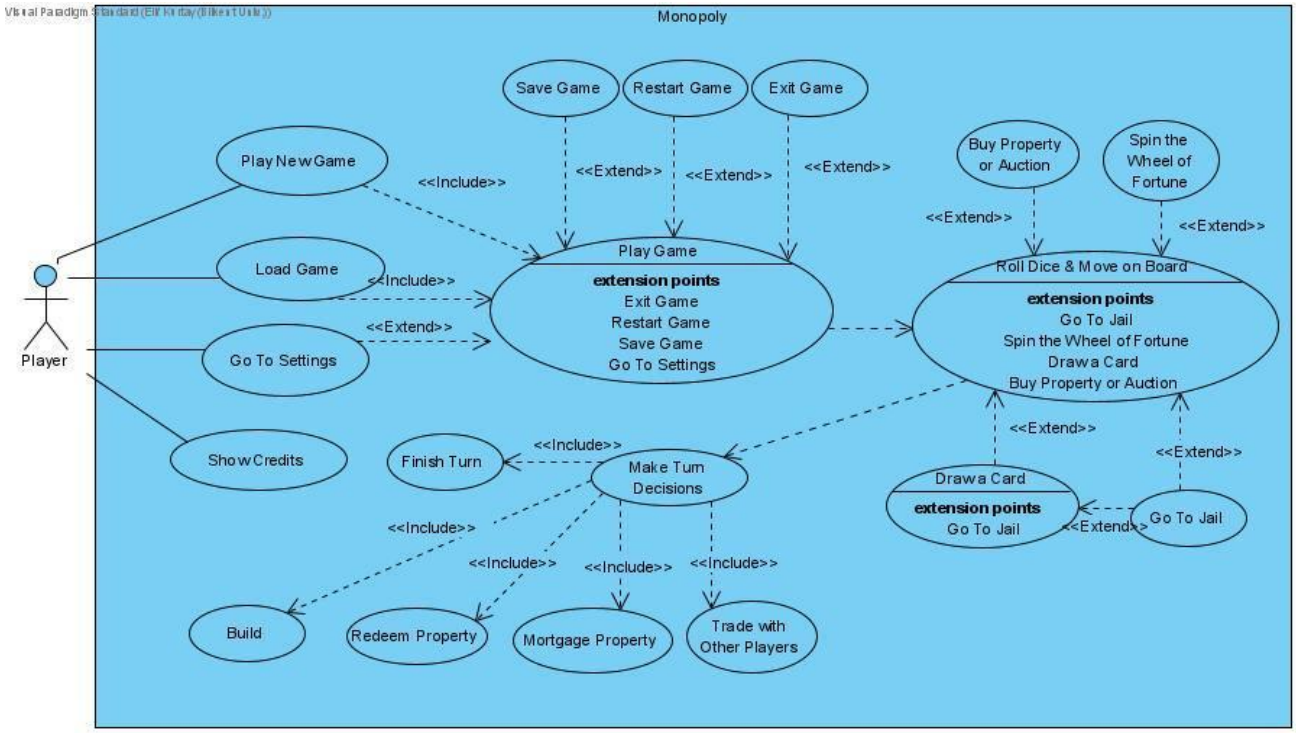

Figure 1. Use case model for the Monopoly game.

Use Case Descriptions

Use Case #1:

1. Name: Show Credits
2. Participating actor: Player
3. Entry condition:
   - Player must be on the main menu.
   - Player must click the “Credits” button on the main menu.
4. Exit condition:
   - Player clicks on the “Back” button on the credits page.
5. Flow of events:
   5.1. Player clicks the “Credits” button on the main menu.
   5.2. System renders the credits page.
   5.3. Player can read the credits.
   5.4. Player clicks the “Back” button when they want to go back to the main menu.
6. Special/quality requirements:
   - No special/ quality requirements

Use Case #2:

1. Name: Go To Settings
2. Participating actor: Player
3. Entry condition:
   - Player is on the main menu or playing the game.
   - Player clicks on the “Settings” button on the screen.
4. Exit condition:
   - Player clicks on the “Back” button on the settings page.
5. Flow of events:
   5.1. Player clicks on the “Settings” button.
   5.2. System renders the settings screen.
   5.3. Player changes the settings that are about adjusting the music and sound volumes.
   5.4. System adjusts the volume as requested from the Player.
   5.5. Player exits by clicking on the “Back” button on the screen.
6. Special/quality requirements:
   - System should have access to the computer's volume.

Use Case #3:

1. Name: Load Game
2. Participating actor: Player
3. Entry condition:
   - Player clicks on the “Load Game” button on the main menu.
   - Player should have previously played and saved games on the same computer.
4. Exit condition:
   - Player clicks on the “Back” button on the screen.
   - Player clicks on the “Yes” button on the prompt for affirmation after selecting a save and clicking on the “Start Game” button.
5. Flow of events:
   5.1. Player clicks on the “Load Game” button on the main menu.
   5.2. System displays the load game selection screen.
   5.3. Player chooses from their previously saved games.
   5.4. System loads the game settings from the chosen save data and requests affirmation from Player to start the game.
   5.5. System starts the game and renders the game screen.

6. Special/quality requirements:
    - Player cannot change the game settings of the loaded game.
    - Player needs to have at least one previously loaded and not finished game saved on their computer.
    - If Player does not have previously saved games, the system should prompt the user to select the "New Game" option in the main menu.

Use Case #4:

1. Name: Play New Game
2. Participating actor: Player
3. Entry condition:
    - Player clicks on the "New Game" button on the main menu.
4. Exit condition:
    - Player clicks on the "Back" button on the screen.
    - Player clicks on the "Start Game" button on the screen.
5. Flow of events:
    5.1. Player clicks on the "New Game" button on the main menu.
    5.2. System renders the game settings screen.
    5.3. Player can add other players and give usernames to all players by pressing on the keyboard. Player also can delete players.
    5.4. Player can choose which map to play on.
    5.5. Player can choose a lap limit.
    5.6. Steps 3, 4, and 5 can be repeated with any order and as many times as wanted by the Player.
    5.7. Player clicks the "Start Game" button on the screen when they are finished adjusting the game settings.
    5.8. System starts the game with the given game settings.
    5.9. System renders the game screen.
6. Special/quality requirements:
    - Player can add a maximum of four players including themselves.
    - The system will display a preview for the selected maps.
    - If less than four players are given, the system will auto-generate players to reach four players.
    - While adding a player, Player needs to select a character for that player.

Use Case #5:

1. Name: Play Game
2. Participating actor: Player
3. Entry condition:
   - Player clicks the “Yes” button on the loading game screen or the “Start Game” button on the game settings screen.
4. Exit condition:
   - System shows the winner when an ending condition is met in the game.
5. Flow of events:
   5.1. System renders the game screen.
   5.2. System requests all Players to roll dice before the game starts to decide on the player's turn order.
   5.3. System starts the first turn of the players and in their turn asks them to choose their character.
   5.4. System starts the turn of a player by enabling Roll Dice and Move on Board use case.
   5.5. Player clicks “Finish Turn” when they no longer want to perform an action in their turn.
   5.6. System decides the next Player and starts their turn.
   5.7. All Players repeat steps 4, 5, and 6 according to their turn order.
   5.8. When all 4 four players complete their turn, System increments the lap counter of the game.
   5.9. The game repeats 7 and 8 steps until one of the ending conditions are met.
   5.10. System shows the winner on the screen and prompts the user to go back to the main menu.
6. Special/quality requirements:
   - At any time during the game, the player can save their progress, quit the game, restart the game, or view the settings.
   - The ending conditions are either reaching the end of their lap limit or having three players go bankrupt.

Use Case #6:

1. Name: Role Dice and Move on Board
2. Participating actor: Player
3. Entry condition:
    - Player begins their new turn in the Play Game use case.
    - Player is in the Game Screen.
4. Exit condition:
    - Player can now perform their Make Turn Decisions use cases.
    - Player will be in a different space on the board if they are not in jail.
5. Flow of events:
    5.1. System renders the Roll Dice Screen.
    5.2. Player clicks on the "Roll Dice" button.
    5.3. System returns a dice number to the Player.
    5.4. System checks if the Player is in jail. If the Player is in jail and either a double is rolled or Player's jail time is completed, Player exits jail. If the Player is in jail and neither a double is rolled nor Player's jail time is completed, Player stays in jail.
    5.5. System returns to the Game Screen.
    5.6. System checks if Go To Jail exceptional case is matched. If it is, 7^{th} step is skipped, and Player's token is moved to the jail space.
    5.7. System performs movement of Player's token as much as the sum of the dice roll.
    5.8. System checks if Go To Jail or Spin the Wheel of Fortune exceptional cases are matched.
6. Special/quality requirements:
    - Result of the dice roll is a computer-generated random value.

Use Case #7:

1. Name: Spin the Wheel of Fortune
2. Participating actor: Player
3. Entry condition:
    - Player is in the Roll Dice and Move on Board use case.
    - Player comes to the "Wheel of Fortune" space on the board.
4. Exit condition:
    - Player returns to Game screen.
    - Player can perform the Make Turn Decisions use case.
5. Flow of events:
    5.1. System loads the Spin the Wheel of Fortune screen.

5.2. Player spins the wheel.

5.3. System randomly selects a price to be given the current Player.

5.4. Player collects their prize.

5.5. System returns to the Game Screen.

6. Special/quality requirements:
    - Result of the price location of the Wheel of Fortune is a computer-generated random value.

Use Case #8:

1. Name: Go To Jail
2. Participating actor: Player
3. Entry condition:
    - Player is in the Roll Dice and Move on Board use case and Player rolls their third doubles.
    - Player is in the Roll Dice and Move on Board use case and Player comes to the "Go To Jail" space on the board.
    - Player is in the Draw a Card use case and Player draws the "Go To Jail Directly" card.
4. Exit condition:
    - Player is in jail.
5. Flow of events:

    5.1. Player's token is moved to Jail space on the board.

    5.2. System puts the Player in jail.
6. Special/quality requirements:
    - Player cannot already be in jail for this use case to be called.

Use Case #9:

1. Name: Draw a Card
2. Participating actor: Player
3. Entry condition:
    - Player needs to be in the Roll Dice and Move on Board use case.
    - Player is on one of a "Draw Chance Card" spaces on the board.
    - Player is on one of a "Draw Community Chest Card" spaces on the board.
4. Exit condition:
    - Player returns to Game Screen.
    - Player can perform Make Turn Decisions use case.

5. Flow of events:
    5.1. Player draws a card either from Community Chest card deck or from the Chance card deck according to their location on the board.
    5.2. System draws a card and prompt the Player if they want to open the card now or one lap later.
    5.3. If the Player wants to open now, the card details are displayed and the action on the card is performed. If the opened card is “Get Out of Jail Free”, the card can be kept in Assets for the during of the game.
    5.4. If the Player wants to save the card for a lap, the card goes to the Assets of the Player.
    5.5. System checks for the Go To Jail exceptional case.
6. Special/quality requirements:
    - Card actions might affect the assets, location of the Player. In addition, the “Thief” card introduces a new fake Player to the game. However, this Thief player does not affect the current Player’s turn.

Use Case #10:

1. Name: Buy Property or Auction
2. Participating actor: Player
3. Entry condition:
    - Player is the Roll Dice and Move on Board use case.
    - Player’s current location is on a property that is in the bank (not owned by any Player).
4. Exit condition:
    - The property is owned by one of the Players in the game.
    - Player can perform Make Turn Decisions use case.
5. Flow of events:
    5.1. System shows the property details to the Player.
    5.2. System prompts the Player between buying the property or starting an auction between other Players to sell the property.
    5.3. If the Player chooses the buy the property, the property is added to their Asset. If the Player chooses to start an auction for the property, System renders the Auction screen and all Players in the game can place their bids for the property or fold from auctioning. The owner of the highest bid at the end of the auction will be the owner of the property.
    5.4. System will update the Assets of the Players and return to the Game Screen.
6. Special/quality requirements:
    - When the property is auctioned to a different Player, the current Player does not pay rent to that Player.

Use Case #11:

1. Name: Make Turn Decisions
2. Participating actor: Player
3. Entry condition:
    - Player completed Roll Dice and Move on Board use case.
    - Player is in the Game Screen.
4. Exit condition:
    - System starts the turn of the next Player.
5. Flow of events:
    5.1. System checks if the Player is in jail or not.
    5.2. If Player is not in jail, Player can perform the Mortgage Property, Redeem Property, Trade with Other Players, Buy Property or Auction, Build, Draw a Card, and Finish Turn use cases. If the Player is in jail, Player can perform the Mortgage Property, Redeem Property, and Finish Turn use cases.
    5.3. System will remain in the Player's turn until Player clicks on the "Finish Turn" button which will initiate Finish Turn use case.

Use Case #12:

1. Name: Trade with Other Players
2. Participating actor: Player
3. Entry condition:
    - Player is in Make Turn Decisions use case.
    - Player clicks the "Trade" button in their Game Screen.
4. Exit condition:
    - Player's turn continues in the Make Turn Decisions use case.
5. Flow of events:
    5.1. System renders "Trade Screen".
    5.2. Player chooses another Player to do trade with.
    5.3. Player in turn and the selected Player fills a trade table displayed by the system. The table shows information of both Player's assets and requires them to choose which items they are willing to trade.
    5.4. After Players reach a consensus on their trading deal, the Player who has the turn clicks on the "Offer" button and other Player clicks on the "Accept" button. If there is no consensus Players can click either on "Cancel" or "Decline" buttons.
    5.5. System updates the Player's asset information.
    5.6. System renders the Game screen.

Use Case #13:

1. Name: Mortgage Property
2. Participating actor: Player
3. Entry condition:
    - Player is in Make Turn Decisions use case.
    - Player clicks on the “Mortgage” button.
4. Exit condition:
    - Player’s turn continues in the Make Turn Decisions use case.
5. Flow of events:
    5.1. System renders Mortgage Screen.
    5.2. System shows all the properties of the Player and their mortgage payments that are not already mortgaged.
    5.3. Player chooses the properties they wish to mortgage or clicks on the “Cancel” button if they no longer wish to mortgage any of their properties.
    5.4. Player clicks on the “Mortgage” button to mortgage.
    5.5. System updates Player’s assets and renders the Game Screen.
6. Special/quality requirements:
    - If there is no available property of the Player, system will return an informative message to the Player stating that they have no possible property to mortgage.

Use Case #14:

1. Name: Redeem Property
2. Participating actor: Player
3. Entry condition:
    - Player is in Make Turn Decisions use case.
    - Player clicks on the “Redeem” button.
4. Exit condition:
    - Player’s turn continues in the Make Turn Decisions use case.
5. Flow of events:
    5.1. System renders Redeem Screen.
    5.2. System shows all the properties of the Player that have been mortgaged and their redeeming price.

5.3. Player chooses the properties they wish to redeem or clicks on the “Cancel” button if they no longer wish to redeem any of their properties.
5.4. Player clicks on the “Redeem” button to redeem their mortgaged properties.
5.5. System updates Player’s assets and renders the Game Screen.

6. Special/quality requirements:
    - If there is no mortgage property of the Player, system will return an informative message to the Player stating that they have no possible mortgaged property to redeem.

Use Case #15:

1. Name: Build
2. Participating actor: Player
3. Entry condition:
    - Player is in Make Turn Decisions use case.
    - Player needs to be on one of their properties of which they own all the properties of the same color.
    - Player clicks on the “Build” button.
4. Exit condition:
    - Player’s turn continues in the Make Turn Decisions use case.
    - Player might have buildings on their properties.
5. Flow of events:
    5.1. System renders Build Screen.
    5.2. Player selects from the properties of the same color and clicks “Next”. Player clicks “Cancel” if they no longer want to build a house or a hotel on top of their properties.
    5.3. System shares details about the selected property.
    5.4. Player clicks on the “Build House” button to increase house number on the property.
    5.5. Player can go back to choosing property screen by clicking the “Back” button.
    5.6. When Player no longer wishes to continue building, Player can click on the “Finish” button.
    5.7. System updates the board, Player’s assets and returns to the Game Screen.
6. Special/quality requirements:
    - Player needs to own all properties of the same color.
    - House number difference cannot exceed “ between all properties of the same number. For example, a second house on a property cannot be build, if the rest of the properties of the same color do not have at least one house.

Use Case #16:

1. Name: Finish Turn
2. Participating actor: Player
3. Entry condition:
    - Player is in Make Turn Decisions use case.
    - Player clicks on the “Finish Turn” button.
4. Exit condition:
    - System starts the turn of the next Player in the Play Game use case.
5. Flow of events:

    5.1. System checks if the Player has bankrupted.

    5.2. If the Player has went bankrupt, this information displayed to the Player.

    5.3. System ends the turn of the current Player.
6. Special/quality requirements:
    - The bankruptcy condition is having negative value in their money field.

Use Case #17:

1. Name: Restart Game
2. Participating actor: Player
3. Entry condition:
    - Player must be playing the game.
    - Player clicks the “Yes” button on the prompt after clicking the “Restart” button on the game screen.
4. Exit condition:
    - System renders the game screen with change on the game data.
5. Flow of events:

    5.1. Player clicks the “Yes” button on the prompt after clicking the “Restart” button on the game screen.

    5.2. System resets the game data to the starting preferences.

    5.3. Player starts playing the game with the initial settings.
6. Special/quality requirements:
    - System needs to prompt the user for affirmation after the user clicks the “Restart” button.
    - System needs to reset lap count, players’ assets, and money.

Use Case #18:

1. Name: Save Game
2. Participating actor: Player
3. Entry condition:
    - Player must be playing the game.
    - Player clicks the “Yes” button on the prompt after clicking the “Save” button on the game screen.
4. Exit condition:
    - System renders the game screen.
5. Flow of events:
    5.1. Player clicks the “Yes” button on the prompt after clicking the “Save” button on the game screen while playing the game.
    5.2. System updates the save data of the current game.
    5.3. System renders the main menu screen.
6. Special/quality requirements:
    - System needs to prompt the user for affirmation after the user clicks the “Save” button.
    - System names the save according to the date and time of the start of the played game.

Use Case #19:

1. Name: Exit Game
2. Participating actor: Player
3. Entry condition:
    - Player must be playing the game.
    - Player clicks the “Yes” button on the prompt after clicking the “Exit” button on the game screen.
4. Exit condition:
    - System renders the main menu screen.
5. Flow of events:
    5.1. Player clicks the “Yes” button on the prompt after clicking the “Exit” button on the game screen while playing the game.
    5.2. System posts the last save of the current game to the database.
    5.3. System renders the main menu screen.
6. Special/quality requirements:
    - System needs to have a connection to the database or give a warning to the Player if there is no connection.
    - System needs to prompt the user for affirmation after the user clicks the “Exit” button.

6.2 Dynamic Models

6.2.1 Activity Diagrams

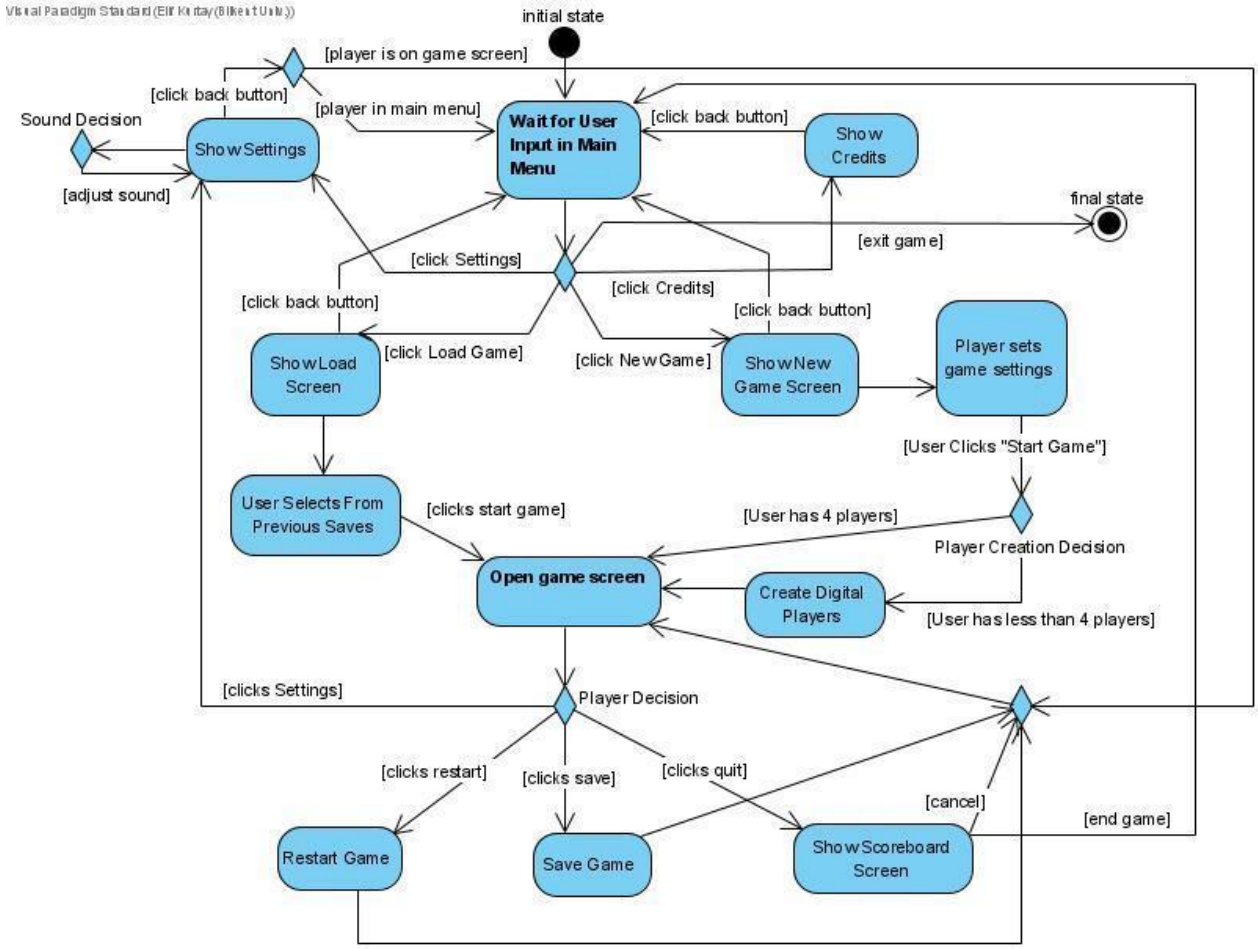

Figure 2. Activity diagram that shows how the system runs the program.

Figure 2 shows the activity diagram that shows how the game runs. The system initializes the program by opening the main menu screen and then waits for user input to give a decision. The player can choose to go to settings for sound and music adjustment, go to the credits page, exit the program, load a previously played game, or start a new game. The player can come back to the main menu after performing any of the actions by clicking the back button.

If the load game choice is selected, the system will gather data from the database and show the load screen to the user. Then the system will wait for a selection from the saved games. After the selection of a saved game, the system will load game data from that save and open the game screen with previous data. If the new game choice is selected, the system will show the new game screen which has multiple sections for different game settings including adding players, choosing a map, and deciding on a lap limit. After the user is done with their desired game setting preferences, the system will wait for a “start game” action from the player. When this action comes, the system needs to decide regarding multiplayer or single-player game mode. If the player entered less than 4 players, the system would create digital “fake” players to complete the player number to

four. Making sure there are four players, the system will render the game screen with default data where lap count is zero.

During the game, each player has the same game decisions in their turns. They can restart the game which resets game history and reduces the lap count to zero. Hence, after the restart, the system goes back to the special sequence for new games. They can save the game which saves game data to the database to be loaded later. They can visit the settings page. They can exit the game, which makes the system show the game over screen with players' scores. The player can cancel their decision to exit the game and go back to playing the game or the system will render the main menu screen (Figure 3).

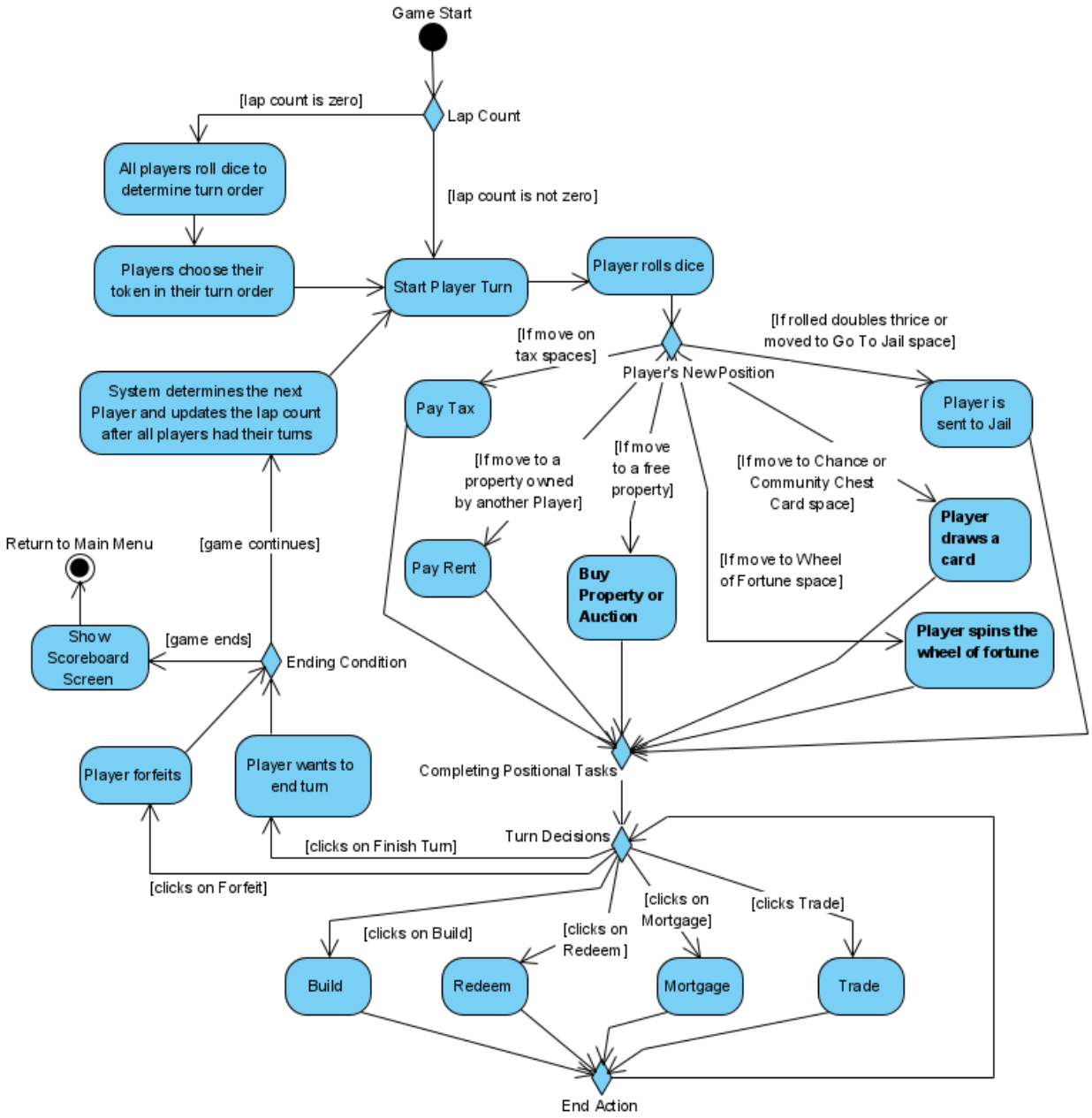

Figure 3. Activity diagram that shows player actions for the turn of a player.

When the game screen is loaded, the system checks lap count. If the lap count is zero, the system starts a special sequence where all players need to roll the dice and then choose their tokens before the game starts. After this sequence completes, the system starts playing the game that is active if the lap count was not zero when the game screen was loaded.

In their turn, the player will roll the dice. Later, the system moves the player according to the dice roll and enables positional tasks for the player. These tasks depend on the position of the player after their dice roll. After completing these tasks, players can select from their turn actions. Turn actions of the player can be trading, building on their property, mortgaging, and redeeming. The turn decisions can be repeated. After the player concludes their turn decisions, they can finish their turn or they can forfeit the game. Whether a user decides to finish their turn or forfeit the game the ending condition will be checked. These conditions are if anyone left playing the game, reaching the lap limit or going bankrupt. If one of the conditions is met, the system will show the game over screen. Otherwise, the system will decide on the next player and restart the “playing the game” loop for that player. If the next player is a “fake” player, then the system will play the default actions for this autogenerated player.

6.2.2 Sequence Diagrams

Scenario 1: Playing the Game

The player wants to play the game. The player enters the main menu and presses the Start Game button. Then the player enters the players, turn limit, and the map of their choice in the New Game Settings screen. The number of players is saved to the *Game* class as a property (attribute) in the *playerCount* variable. After the number of players and digital players are determined, the *Game* class initializes the game and creates the Board, the Bank, and The Event Handler. Then the game loop starts. The Board places the characters on the board, randomizes the drawable cards, sets the initial volume, and other settings. Players' attributes determine whether a player can roll the dice on that turn or not. Similarly, the operations of the characters that the players possess determine how the characters interact with each other. In any turn, the player can save, restart, and exit the game (see Figure 4).

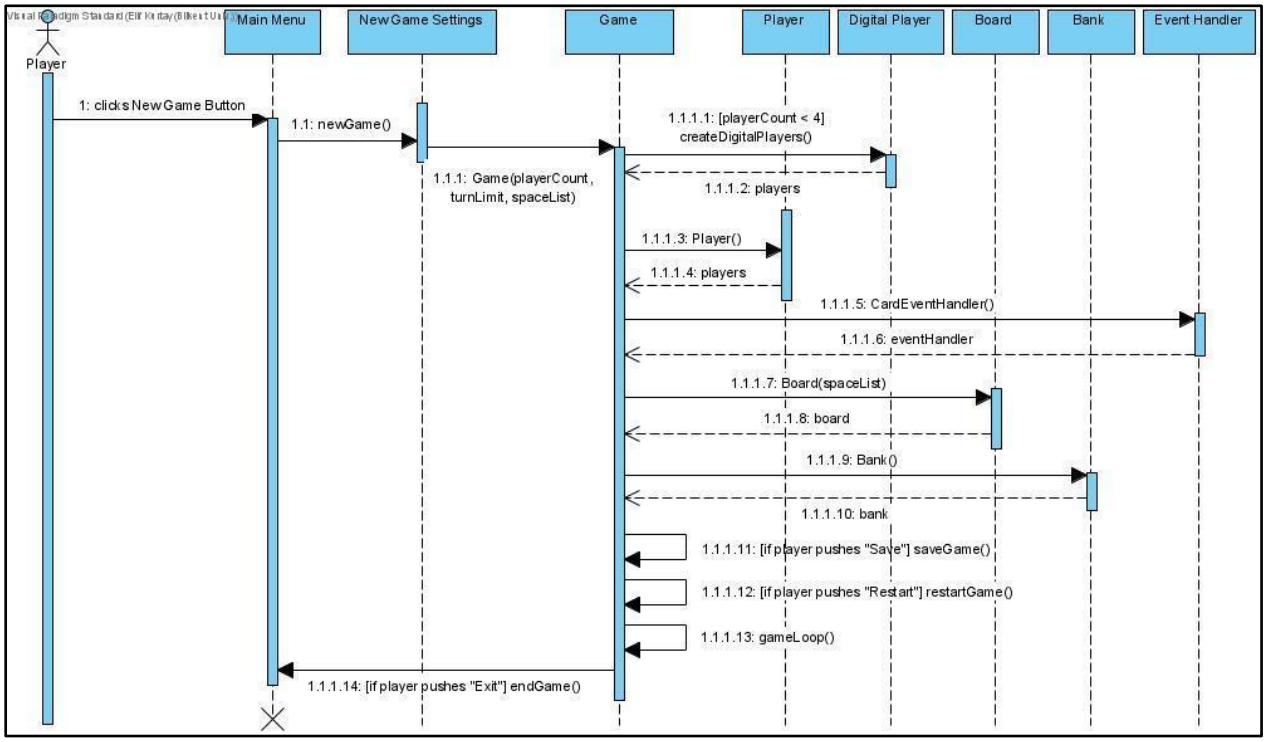

Figure 4. Sequence diagram for playing the game.

Scenario 2: Auction

The player lands on a property. If the property is owned the player pays the owner of the property the rent. Otherwise, the player is prompted to buy the space. The player can choose to buy the space by paying the amount to the Bank. The player can also choose to not buy by pushing the "Cancel" button which then starts the auction process. Bank receives the "Cancel" input and initializes the Auction, stores it as a property and attaches itself to the Auction's observers list. The Auction sets its state as 1 (ongoing auction state). The bank starts the auction and Auction changes the Bank's onGoingAuction property to true. Every time a player edits and enters their bid Auction class updates the bids. If all but one players fold auction is closed and

onGoingAuction becomes false. For this specific scenario Player1 and Player2 makes their bids, then Player3, Player4 and Player1 folds to leave Player2 as the winner. Auction sets its state as 0 (finished auction state) and notifies its observers. The bank gets the highest bidder and its associated bid amount and updates itself. The bank appends the auctionedProperty to the highest bidder's properties. The highest bidder (which is Player1 in this scenario) pays the bank the bid amount (see Figure 5).

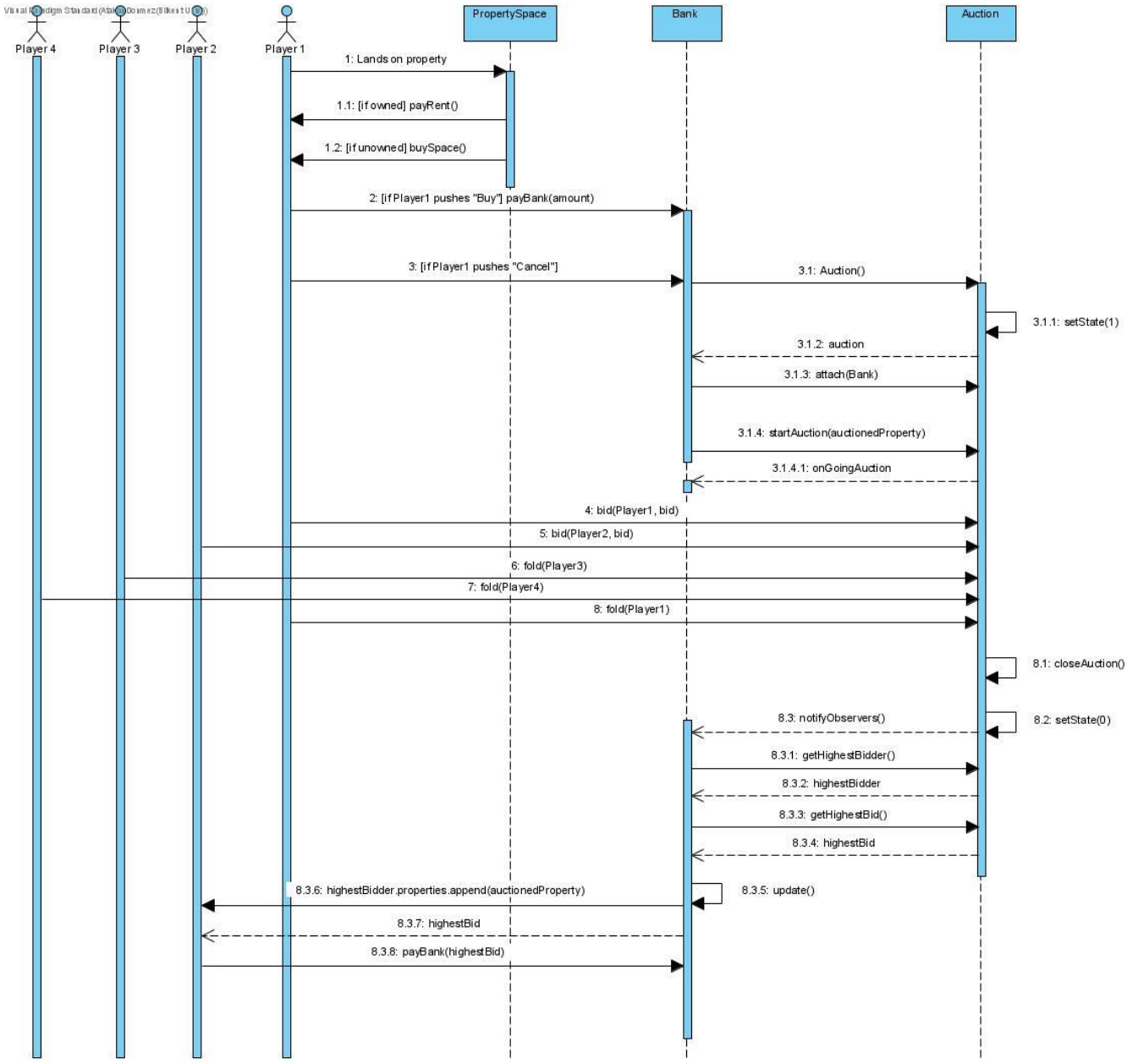

Figure 5. Sequence diagram for auction.

Scenario 3: Deploying Thief

The player lands on a CardSpace. A card gets drawn and player can choose to postpone it. When the card is opened CardEvent is decided. Card class stores the CardEvent class as a property and opens the card. Board class checks if the cardEvent property is an instance of ThiefEvent class and initializes a Thief. The Board class then deploys the thief (see Figure 6) .

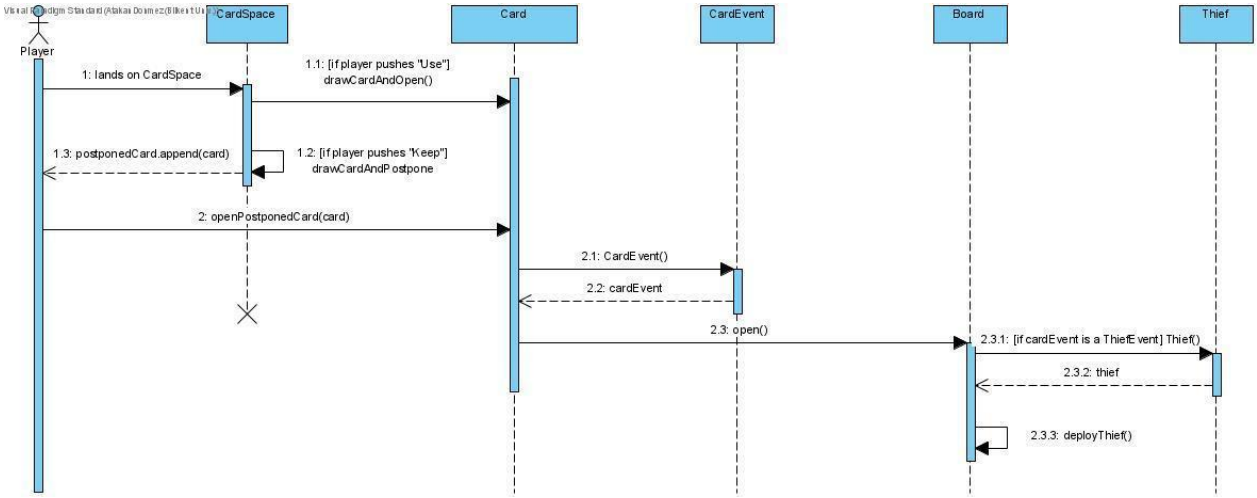

Figure 6. Sequence diagram for deploying the thief.

Scenario 4: Trade

Player1 wants to trade. Player1 clicks the “Trade” button and the Bank class initializes an instance of Trade and starts a trade. The status of the trade is stored in the onGoingTrade property of the Bank class. Each time Player1 edits the offer Trade calls the offer and want methods. Player1 pushes “Cancel” to close the trade or pushes “Offer” to send the final offer. Player2 pushes “Decline” to close the trade or pushes “Accept” to alert the Trade class. The Trade class alerts the Bank to finalize the deal and Bank changes the money, properties and getOutOfJailFreeCount of both players according to the deal (see Figure 7).

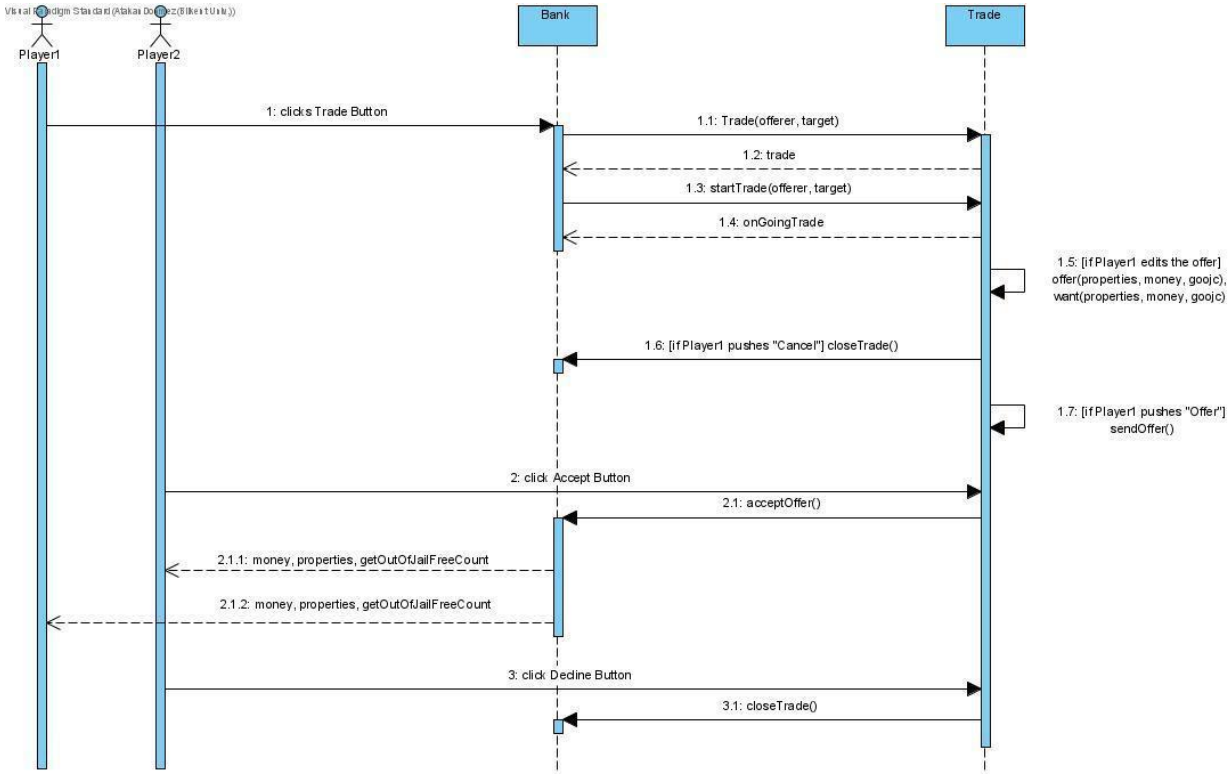

Figure 7. Sequence diagram for trading.

6.2.3 State Diagrams

Player Object's State Diagram

Figure 8 shows the Player state diagram. Players are either created or loaded from previous saves. When a new Player is created, it needs the initializing lap to get a token. After receiving its token, it will be active whereas saved players already have a token, therefore they are active after they are created. Active players get changed during the game when their properties, money, cards, or position on the board gets updated. When jailed, players have limited access to game functions. Players need to be released to be active again. Either inside active or jailed states, if the player has negative balance at the end of their turn, they go bankrupt. However, one player going bankrupt does not mean the game ends. Game can continue while one player is bankrupt. A player can also forfeit when they are active. The game can end when the player is jailed, active, bankrupt or forfeited. After the game ends, Player enters its final state and gets destroyed.

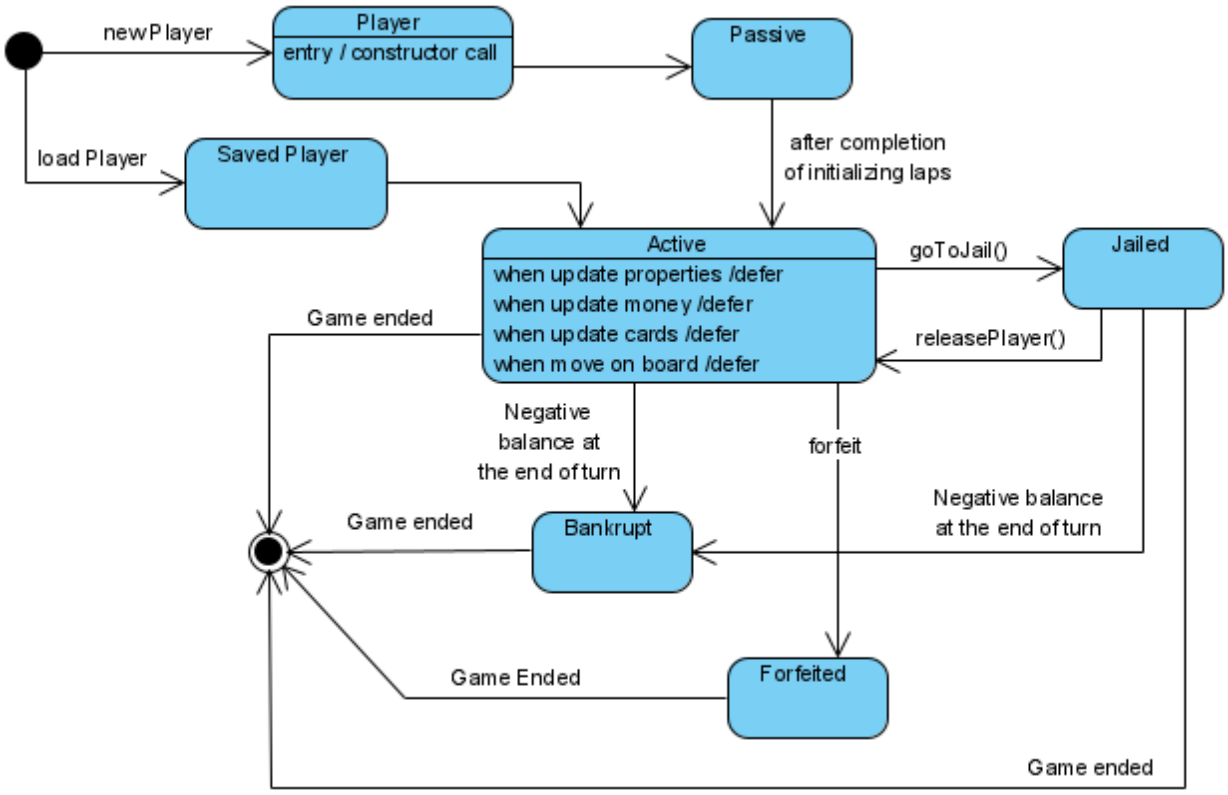

Figure 8. Player state diagram.

Property State Diagram

Properties are created with the Game object. Initially all properties are owned by the bank and only "buy or auction" action can be performed on them when a player comes to the property space. After a property is "owned" by a player, rent can be collected from the property, and property can be traded, mortgaged, and a house can be built upon it. If an owned property is mortgaged, then it is still owned by a player, but it can only be redeemed or traded. When redeemed, the property returns to its owned state. When a property in an owned

state has a hotel built to it. It comes to a new state, called “has hotel”. This state is the same with the owned state except for the build function. It is not allowed to build on top of a hotel. The object gets destroyed when the game ends in any state (see Figure 9).

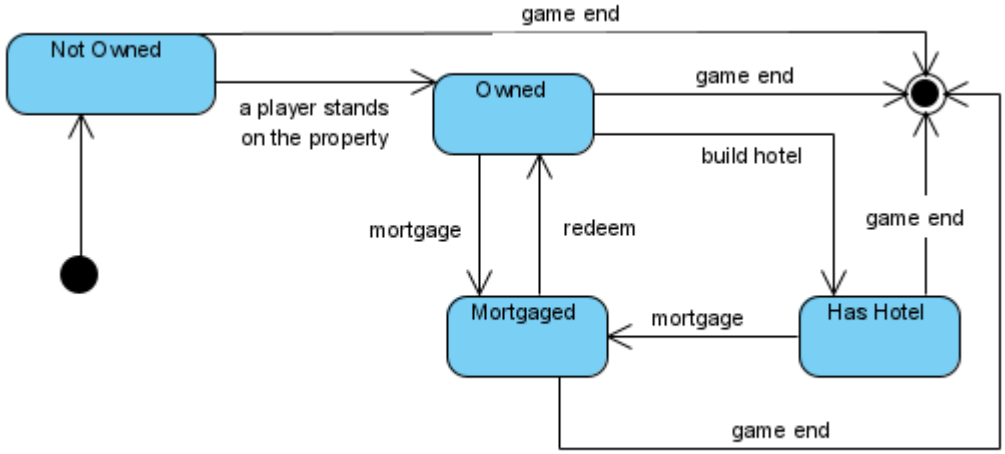

Figure 9. Property state diagram.

Trade State Diagram

Figure 10 shows the trade state diagram. A player can initiate a trade by pressing the trade button when it is their turn. The initial state of the trade action is “Player Selection”. The next state is the “Trading” state. When the users interact with the UI, the table will change. If they press the back button, the next state will be the “Player Selection”. If they press the cancel button, the next state will be the final state of the diagram. After an offer is made, the trade is in the “Offered” state. If it is accepted then the next state will be the final state and the trade will be closed. If it is declined, a new offer can be made or it can be canceled. If a new offer is made the next state will be the “Trading”. If the trade is cancelled in this state then the next state will be the final state and it will be closed.

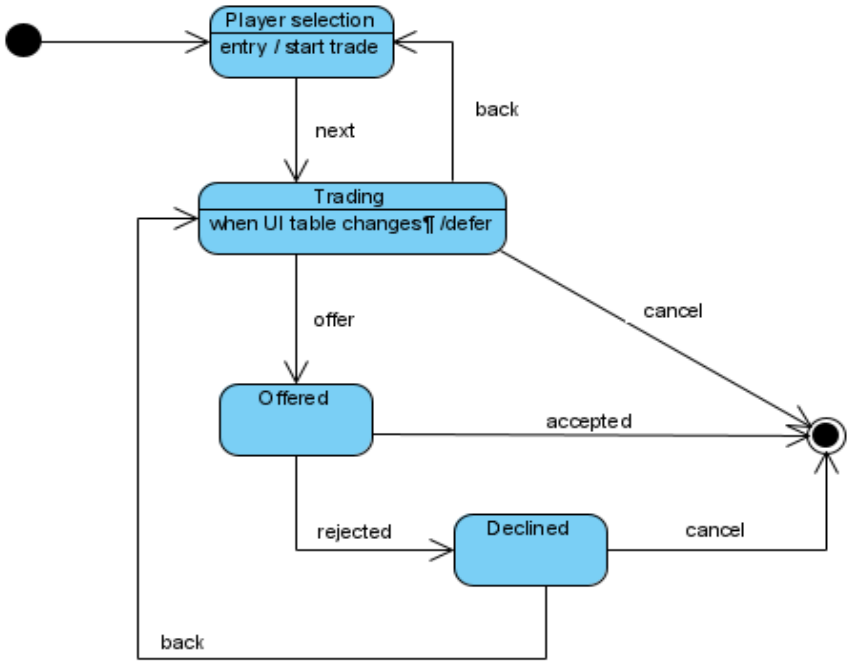

Figure 10. Trade state diagram

Thief State Diagram

Thief player is created when a player draws and uses the thief card. Then, the thief is constructed. After creation, each lap the thief rolls dice and moves on the board until it is on the same space with another player. When thief "catches" a player, it enters the stealing state and steals the money of the player. Then thief gets destroyed by the system (see Figure 11).

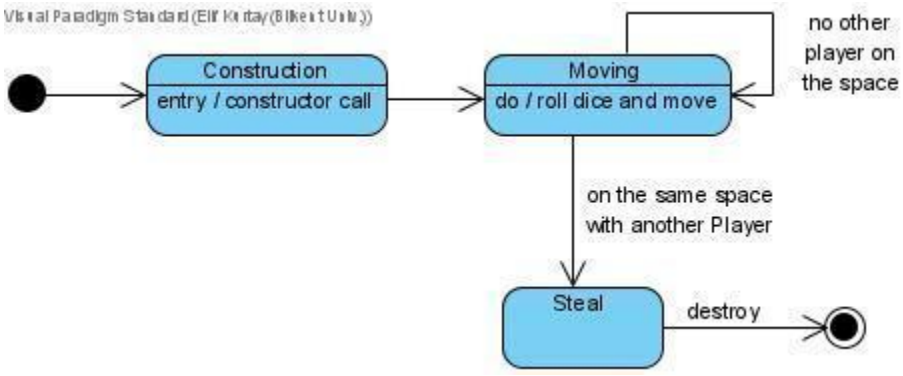

Figure 11 Thief state diagram

Game State Diagram

Figure 12 shows the game state diagram. Game object is either created as a new object or is loaded from previous saves. When it is a new game, game object needs to enter initializing laps to determine player turn order and their tokens. After completing initializing laps, game enters the in-game state. When game is loaded from saves, it automatically enters the in-game state. In this state, players actions make the game update its attributes all the time. The restart command taken during the in-game state returns the game object to initializing laps while resetting information of the object. Either when game ends or is terminated by the players with the exit game command, the game object enters its final state and is destroyed after that.

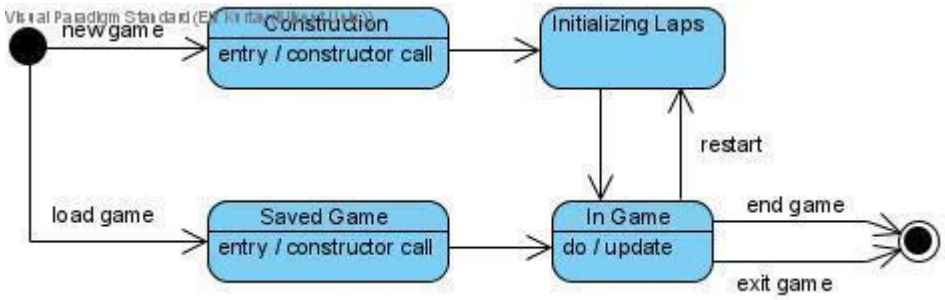

Figure 12 . Game state diagram

6.3 Object and Class Model

Object and Class Model without the UI classes are provided in Figure 13. The descriptions of the classes are as follows:

- *MainMenu Class:* The MainMenu class is our main UI class that is used for displaying the options New Game, Load Game, Settings, Credits, Quit, their subsequent screens, the board, and the UI of the game after the game starts.

- *Game Class:* The Game class is the main class of our game logic. It stores the current player, the current turn number, the number of players, the turn limit if there is a turn limit and consists of the game board, the players, the bank, and the card event handler.

  The Game object can be constructed from a playerCount, turnLimit, and a spaceList as a new game, or from a Game object that was saved before.

  Our game logic will be implemented within the gameLoop() function. The dice rolls, turns, bank operations, player interactions with the board and the other players and ending conditions of the game will be handled within this function.

- *Player Class:* The Player class represents each of the players of the game. It holds values such as the player name, their current money, the space they are currently in, if they are bankrupt or not, the properties they own, how many “Get Out Of Jail Free” cards they own, if they are jailed or not, how many Chance or Community Chest cards they have postponed and haven’t opened yet and which player token they are using.

- *Digital Player Class:* The DigitalPlayer class inherits from the Player class and represents the players that are controlled by the computer. All their actions and responses to activities such as trades and auctions are determined by the computer.

- *Token Class:* The Token class represents each of the tokens the players can choose when starting the game. Each token has different advantages and disadvantages. The token class stores the values that are used for the calculations of these advantages and disadvantages.

- *Property Class:* The Property class represents each property a player can own. Each property has its own TitleDeedCard that stores its rent, or the values used in calculating its rent and the cost of the building. It stores values such as the number of houses built on the property, if there is a hotel on it, and if the property is mortgaged or not.

- *TitleDeedCard Class:* Each property in the game has its own TitleDeedCard. A TitleDeedCard must have the name and the mortgage value of the property. There are 3 child classes that inherit from the TitleDeedCard class, which are UtilityTitleDeedCard, TransportTitleDeedCard, and LandTitleDeedCard classes.

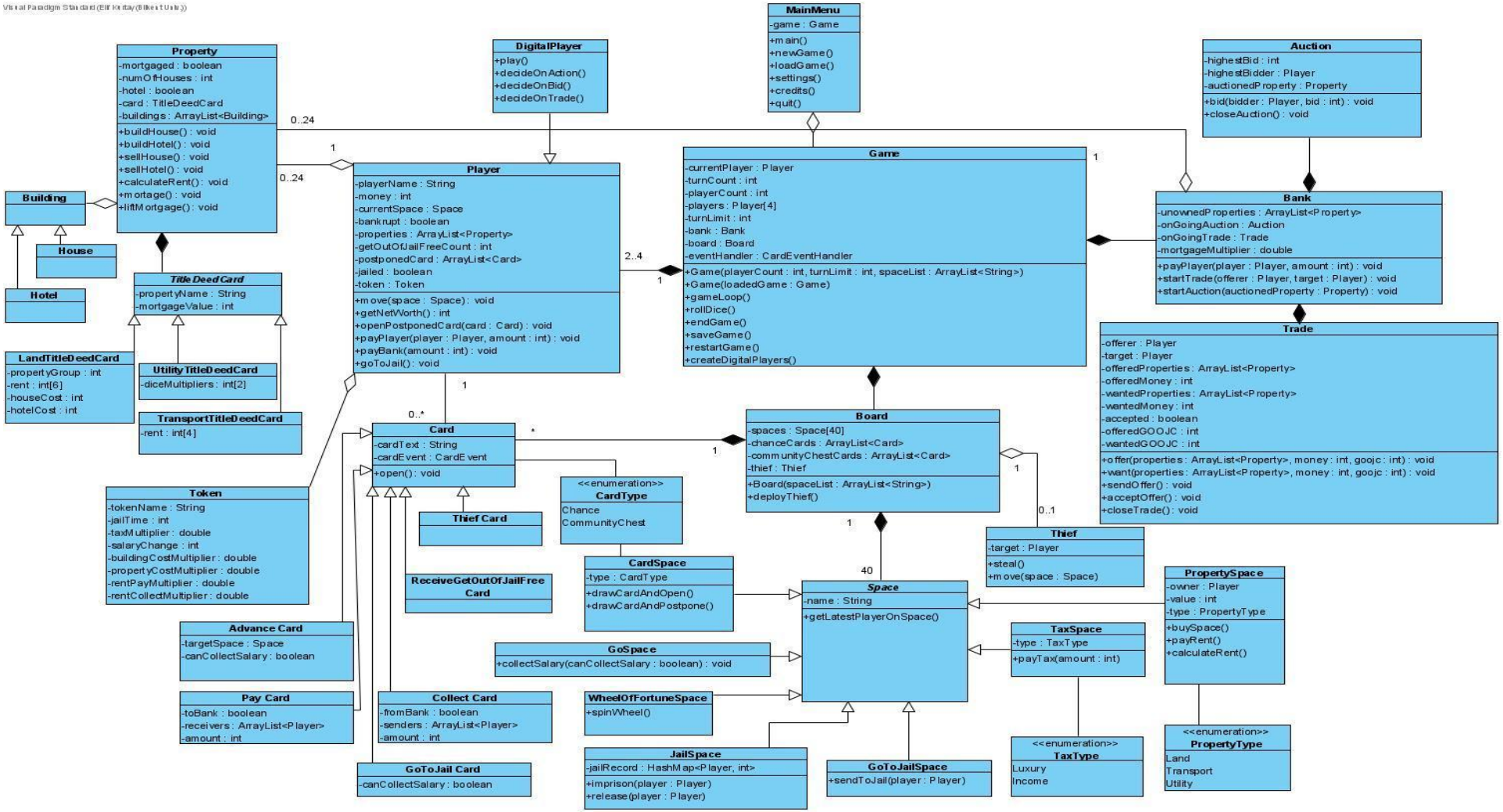

Figure 13. Object and Class Model without the UI classes.

- *LandTitleDeedCard* Class: The LandTitleDeedCard class inherits from the TitleDeedCard class. It holds set values about a land property such as the property group (color group) it belongs to, the rent for each possible state (empty property to having a hotel on it.) of the property, the cost of building houses, and a hotel on it.

- *TransportTitleDeedCard* Class: The TransportTitleDeedCard class inherits from the TitleDeedCard class and holds the rent values for a transport property.

- *UtilityTitleDeedCard* Class: The UtiliyTitleDeedCard class inherits from the TitleDeedCard class and holds the dice multipliers for calculating the rent values for a utility property.

- *Card* Class: The Card class represents the Chance and Community Chest cards from the classic Monopoly game. Each card has a card event, a card text that explains the card event, and has a type which is an enumeration that is either Chance or CommunityChest.

- *CardEvent* Class: The abstract CardEvent class represents events that can occur as a result of opening Chance or Community Chest cards. Each CardEvent has an affected player who is the one who opened the card. The *AdvanceEvent*, *PayEvent*, *CollectEvent*, *GoToJailEvent*, *ReceiveGetOutOfJailFreeCardEvent* classes are the subclasses of the CardEvent class.

  - *AdvanceEvent* Class: An AdvanceEvent advances the affected player to a target space and specifies if the player can collect their salary if they pass through the GO space (the starting space) while advancing to the target space.

  - *PayEvent* Class: A PayEvent makes the affected player pay a certain amount of money to the bank, to a player or players.

  - *CollectEvent* Class: A CollectEvent makes the bank, a player or players pay a certain amount of money to the affected player.

  - *GoToJailEvent* Class: A GoToJailEvent sends the affected player to the jail and specifies if the player can collect their salary if they pass through the GO space (the starting space) while advancing to the jail space.

  - *ReceiveGetOutOfJailFreeCardEvent* Class: A ReceiveGetOutOfJailFreeCardEvent increases the amount of "Get Out Of Jail Free" cards the affected player has by 1.

  - *ThiefEvent* Class: A ThiefEvent deploys a thief to the board, that randomly selects a target player to steal from.

- *CardEventHandler* Class: The CardEventHandler handles the random events that can occur as a result of opening Chance or Community Chest cards. It stores the list events that are not yet handled to handle the case of multiple events happening in one turn, and handles them in the same order they happened.

- *Bank* Class: The Bank class pays the players, holds the unowned properties, manages trades between the players, and conducts an auction if a player decides not to buy a previously unowned space.

- *Auction* Class: The Auction class holds the highest bidder, the highest bid, the auctioned property, and allows the players to bid on the property. When the auction ends the transfer of property is handled by the bank.

- *Trade* Class: The Trade class allows the players to make trades involving money, properties, and "Get Out Of Jail Free" cards. A trade is initiated when a player sends their offer, composed of what they are willing to give and what they want from the opposite side. If the opposite side agrees, the trade is closed and the transfer of assets is handled by the bank.

- *Board* Class: The Board class consists of 40 spaces, has a list of Chance cards, a list of Community Chest cards, and may have a thief. The board is constructed at the beginning of the game from a list of spaces, which may be the default space set of classic Monopoly or a user-defined custom space set.

- *Space* Class: The abstract Space class represents each of the 40 spaces on the classic Monopoly board. *GoSpace*, *WheelOfFortuneSpace*, *GoToJailSpace*, *JailSpace*, *PropertySpace*, *TaxSpace*, and *CardSpace* are all subclasses of the abstract Space class.

    - *GoSpace* Class: The GoSpace class inherits from the Space class and pays salary to players when they go over it if they are not prohibited from receiving a salary.

    - *WheelOfFortuneSpace*: The WheelOfFortuneSpace class inherits from the Space class. It replaces the "Free Parking" space from the Monopoly board. If a player lands on a WheelOfFortuneSpace they will spin a wheel and gain a random property, money, a house on one of their properties if possible, a "Get Out Of Jail Free" card or they may lose money or one of their buildings on one of their properties.

    - *CardSpace*: The CardSpace class inherits from the Space class and has a type which is an enumeration that is either Chance or CommunityChest. If a player lands on a CardSpace they can either draw and open a card from the given type or draw and postpone opening the card.

    - *TaxSpace*: The TaxSpace class inherits from the Space class and has a type which is an enumeration that is either Luxury or Income. If a player lands on a TaxSpace they will have to pay a tax of an amount decided from the type of the TaxSpace.

    - *PropertySpace*: The PropertySpace class inherits from the Space class and has an owner, value, and a type, which is an enumeration that is either Land, Transport, or Utility. If the player lands on an unowned PropertySpace they will have the chance to buy the property associated with that PropertySpace, if they decide not to, the property will go on auction through the bank. If the player lands on a PropertySpace that is not theirs and is owned, they will have to pay rent to the owner of the associated Property.

    - *GoToJailSpace* Class: The GoToJailSpace class inherits from the Space class. If a player lands on the GoToJailSpace, they will be immediately sent to jail and will not be able to collect their salary even if they pass through the GoSpace while advancing to the JailSpace.

- *JailSpace* Class: The JailSpace class inherits from the Space class. The players that are sent to jail via either the GoToJailSpace, a Chance card, or by rolling double 3 times are sent to this space. It holds a HashMap<Player, int>, which stores the number of turns that each player currently in jail has spent in jail.

- *Thief* Class: The thief will be deployed on the board after the handling of a ThiefEvent from a card. The thief will select a random player to target upon its deployment and will move towards them every turn. When the thief catches its target, it will steal a random amount of money from the target.

6.4 User Interface - Navigational Paths and Screen Mock-ups

In this section, we provide the screen mock-ups for the Monopoly game. Please see Section 3. Functional Requirements for the navigational paths between these screen mock-ups, Section 6.1 Use Case Models for various use-case scenarios based on these screen-mockups, Section 6.2.1 Activity Diagrams that shows the flow of the monopoly game in terms of the game activities, Section 6.2.2 Sequence Diagrams the events that players generate, the order of these events, possible system events for specific use-case scenarios, Section 6.2.3 State Diagrams for representing the players' behaviors using finite state transitions. The list of screen mock-ups are as follows.

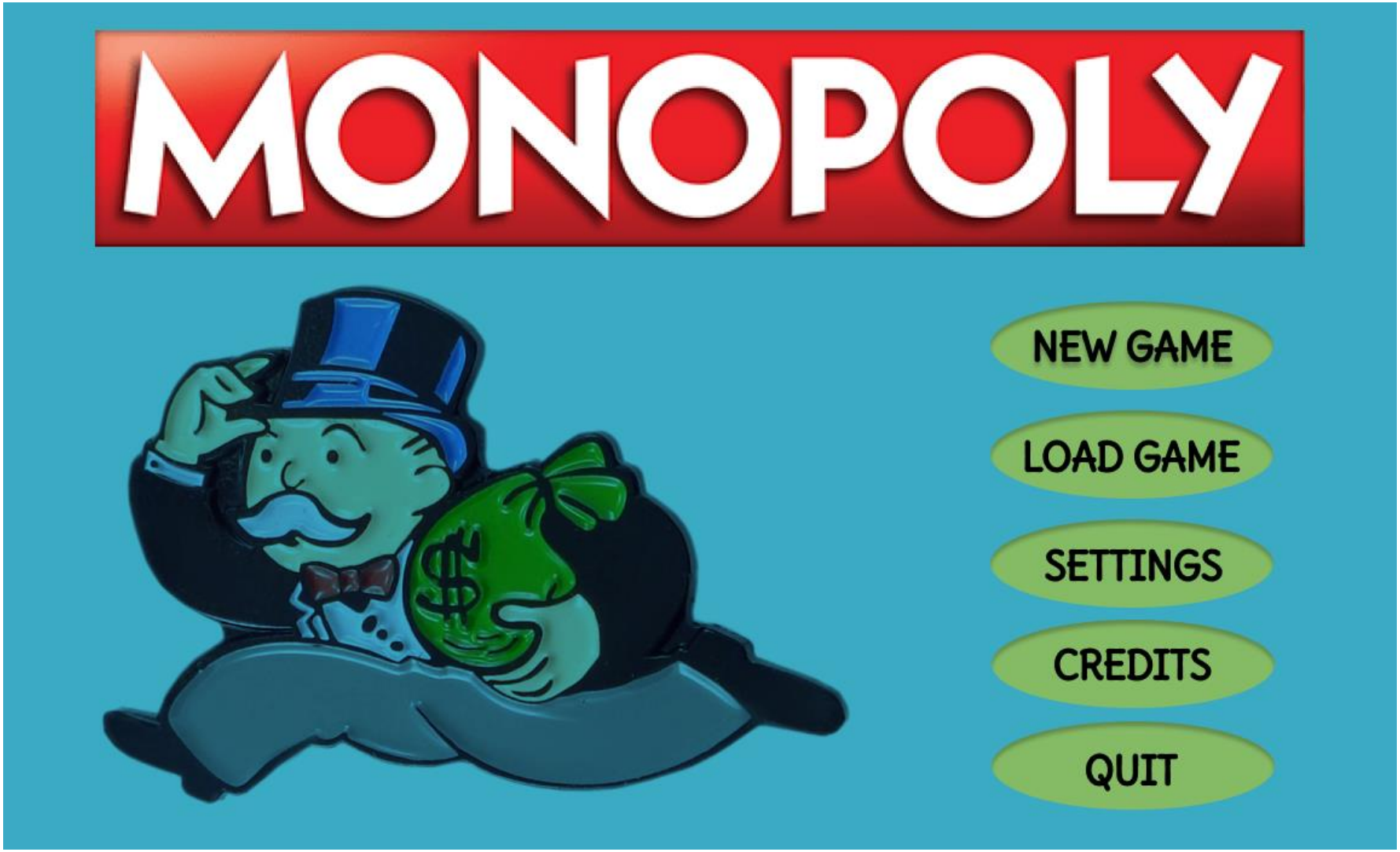

Figure 14. Monopoly game start screen.

Figure 15. Load game screen.

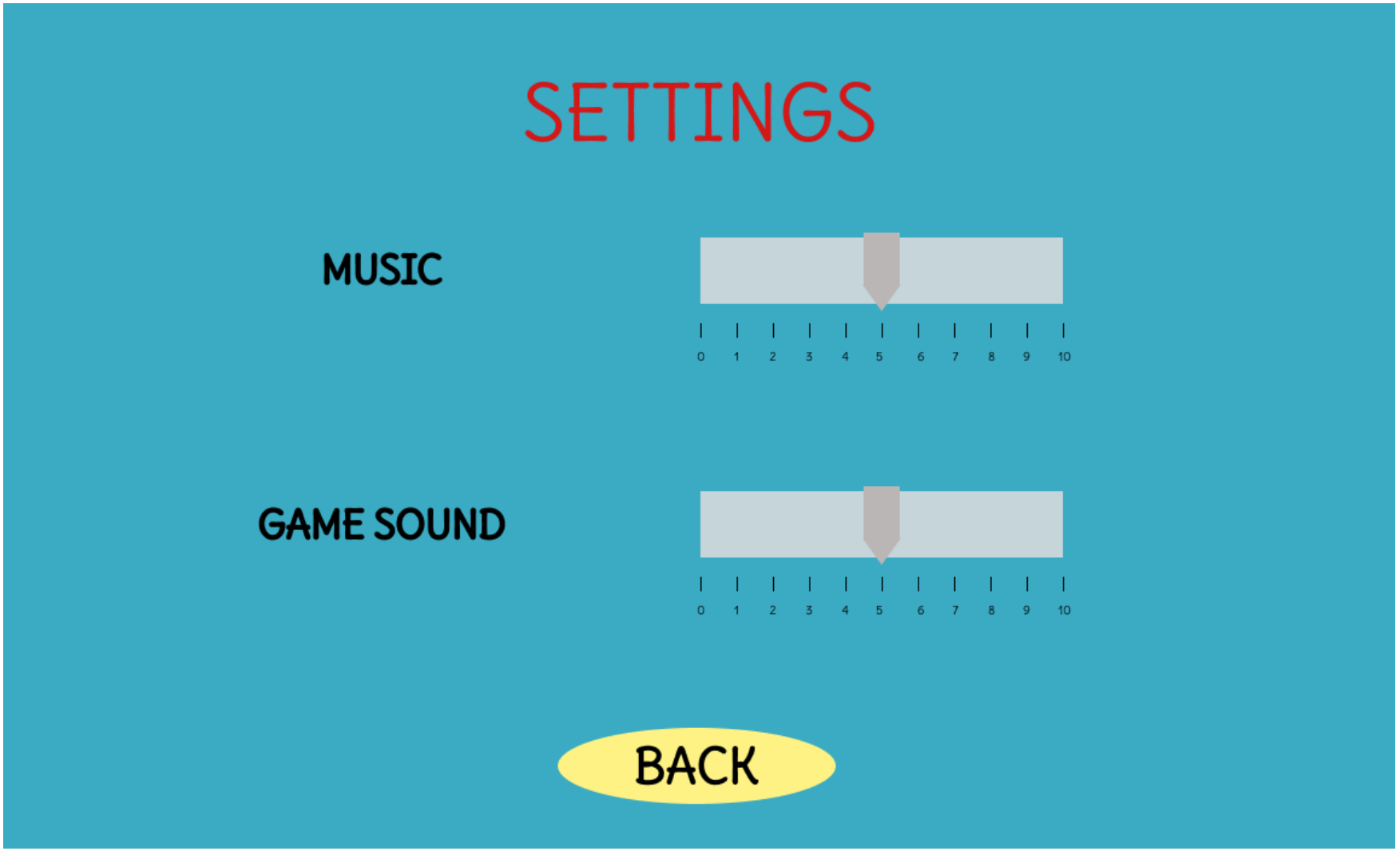

Figure 16. Settings screen.

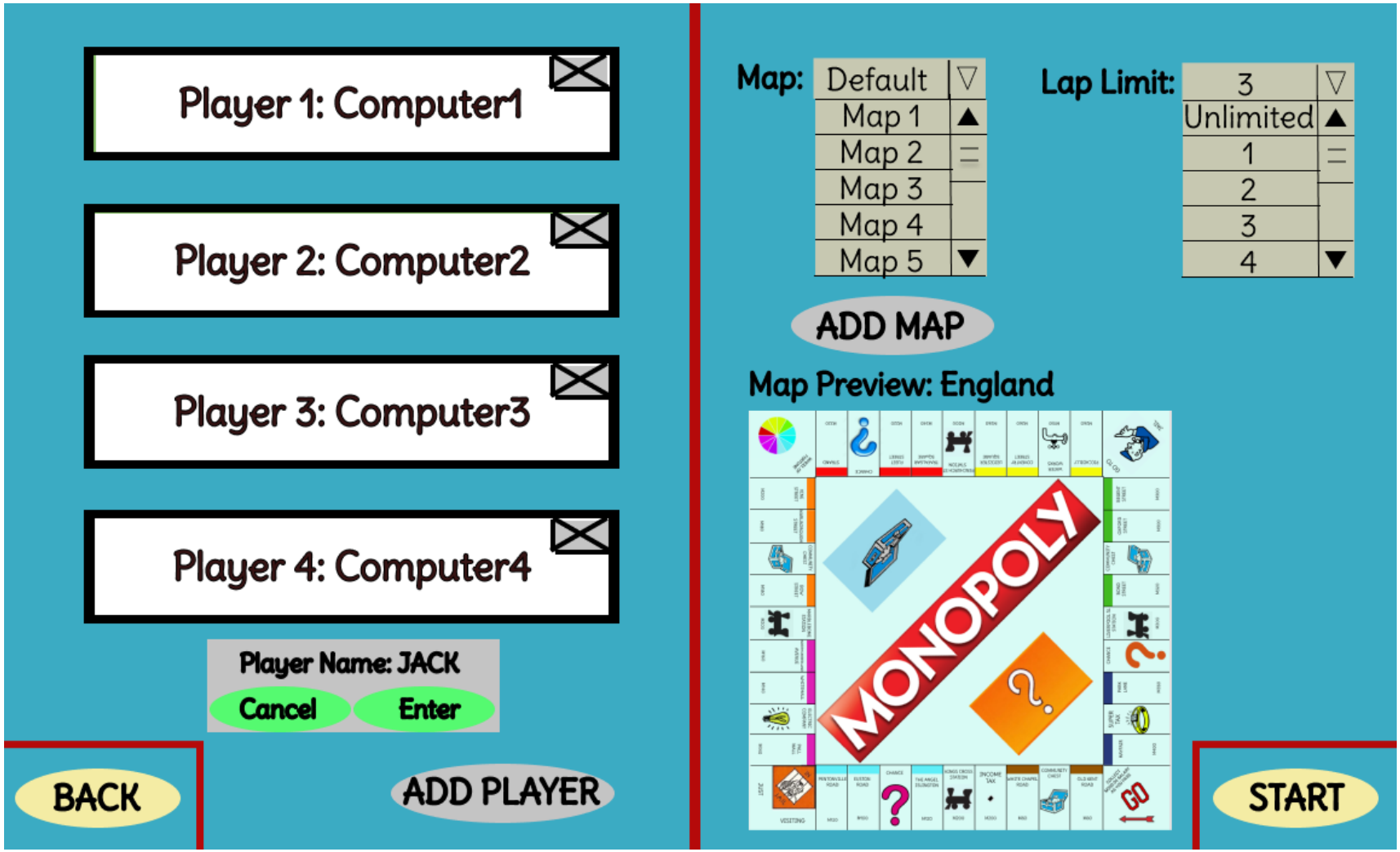

Figure 17. Game lobby offline add player screen.

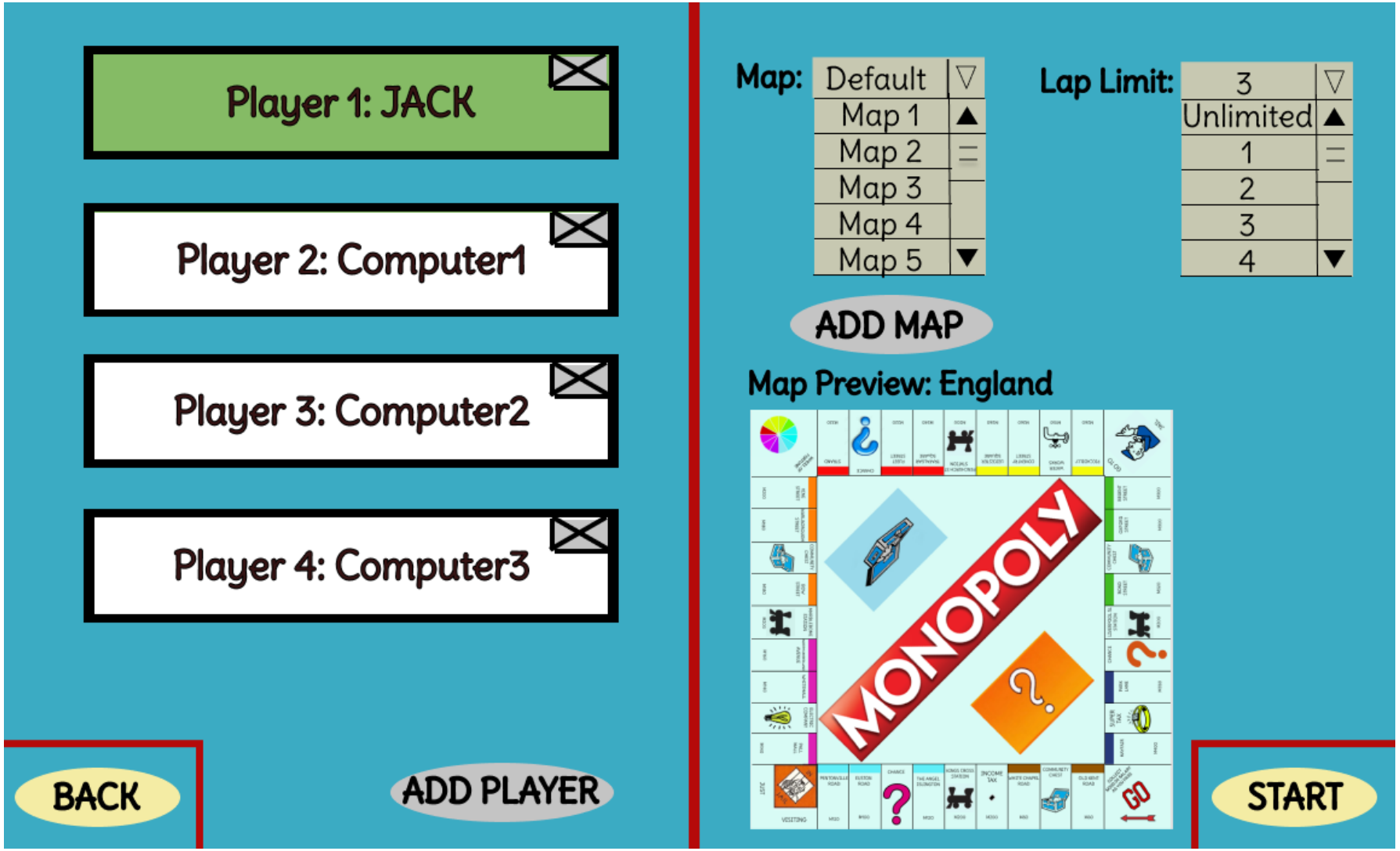

Figure 18. Game lobby offline with players.

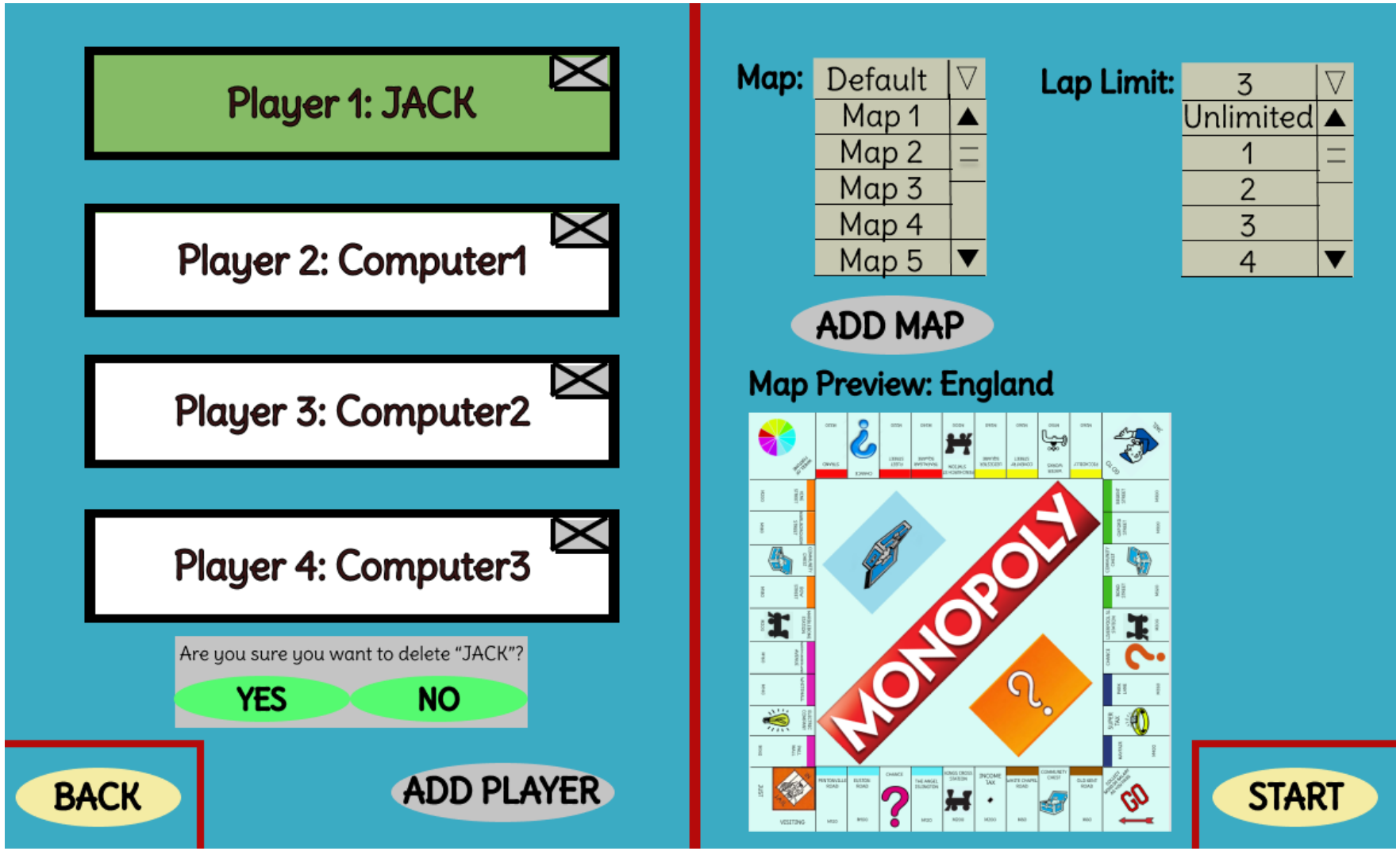

Figure 19. Game lobby offline delete player screen.

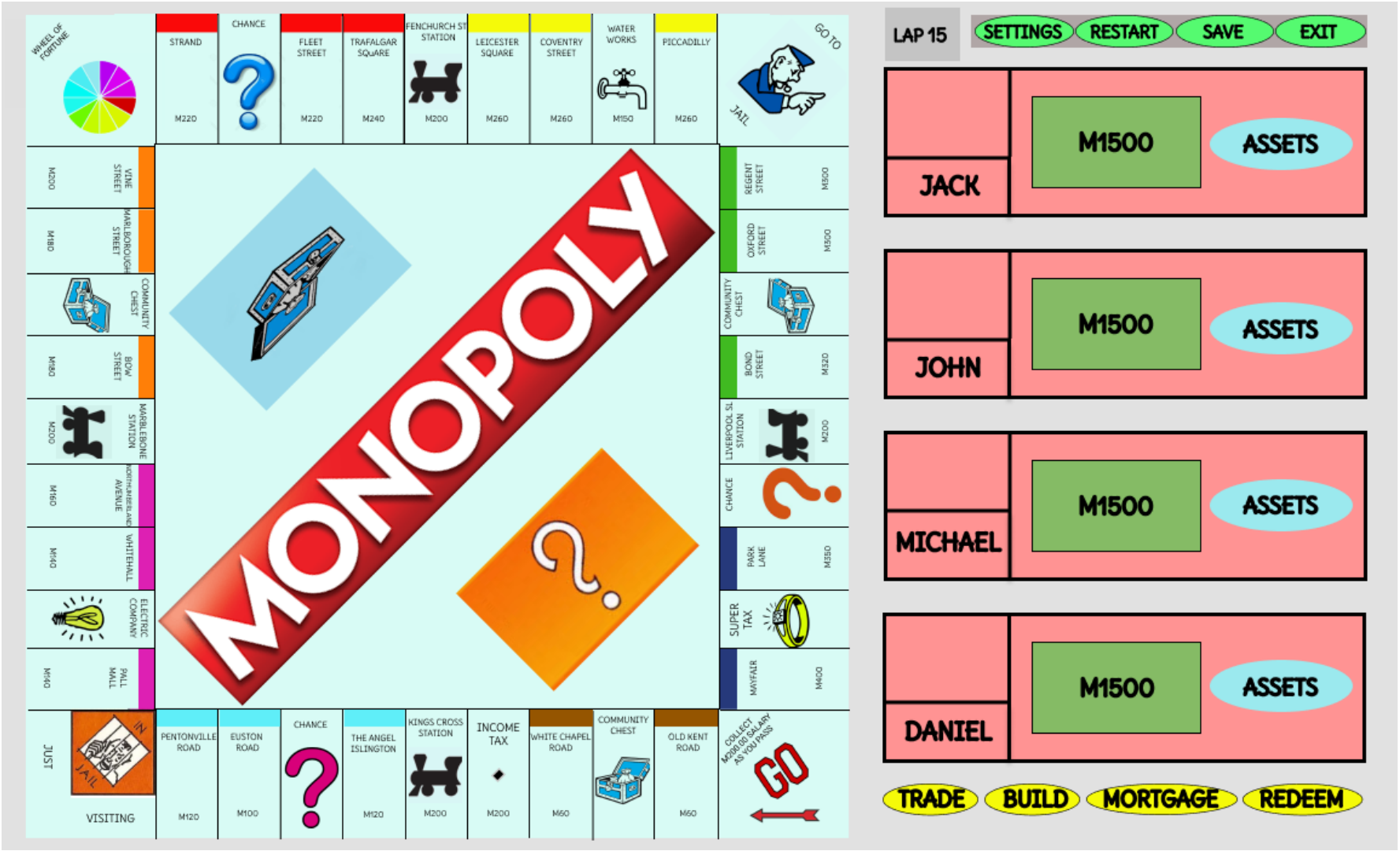

Figure 20. Gameplay starting screen.

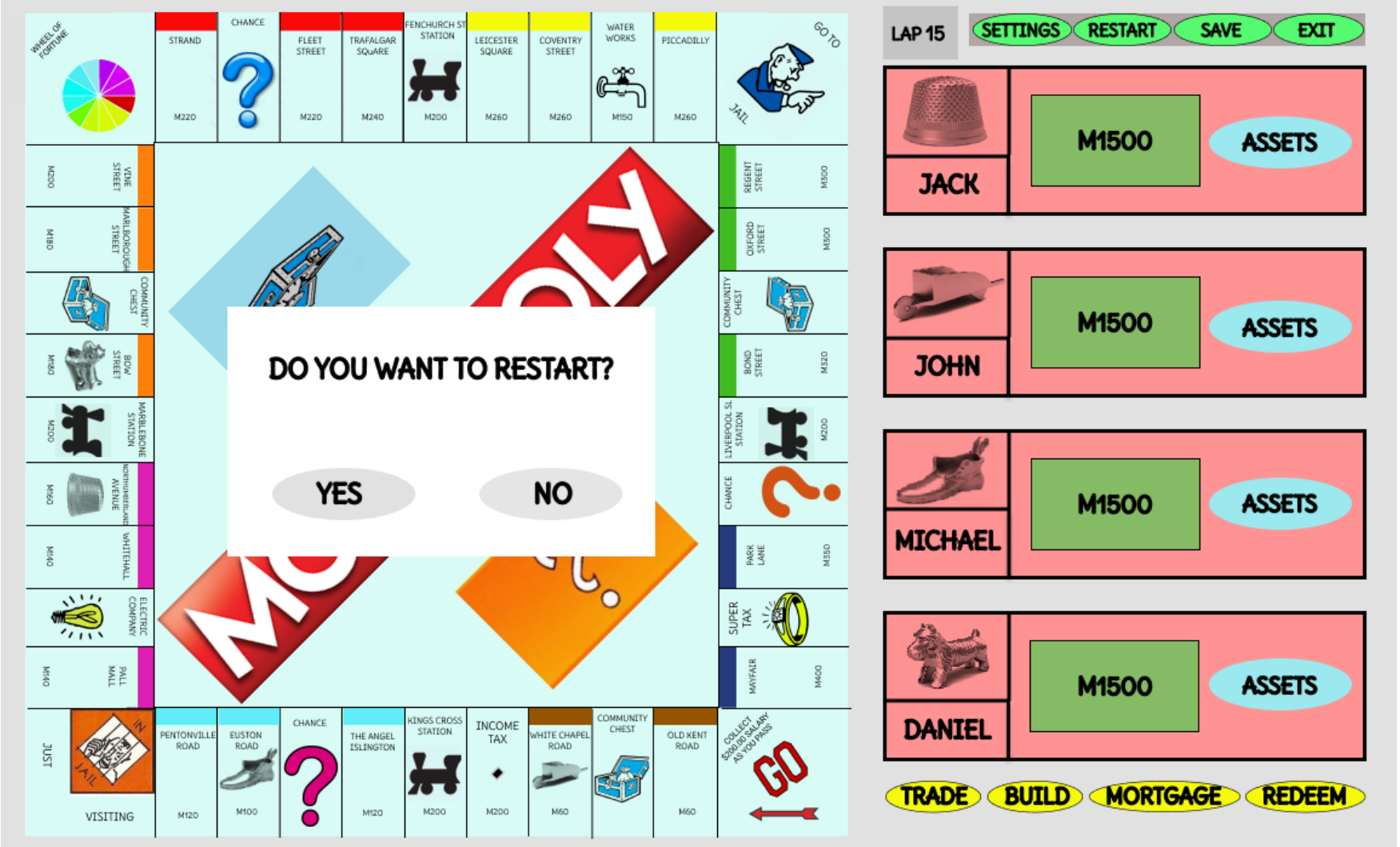

Figure 21. Gameplay restart screen.

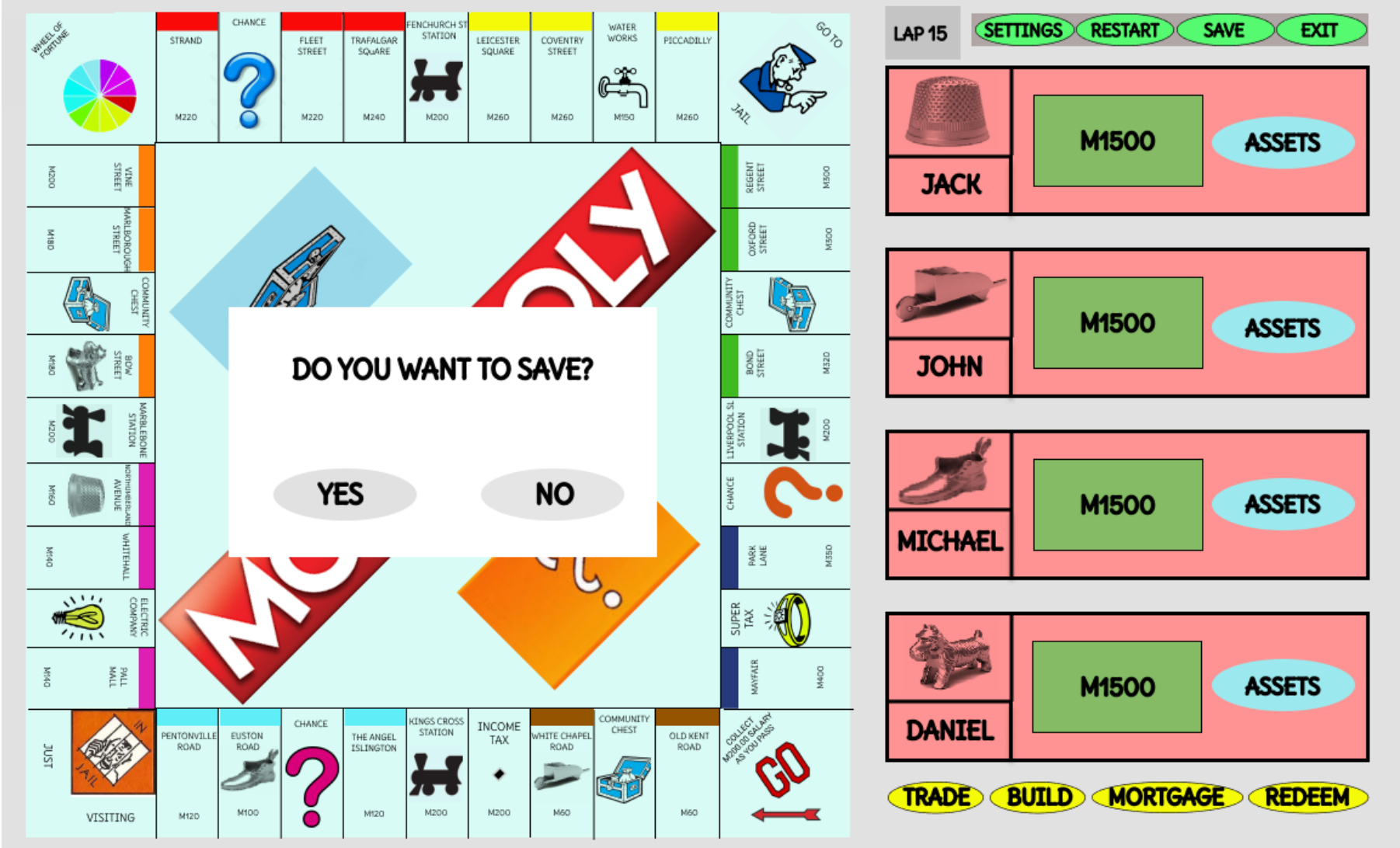

Figure 22 Gameplay save screen.

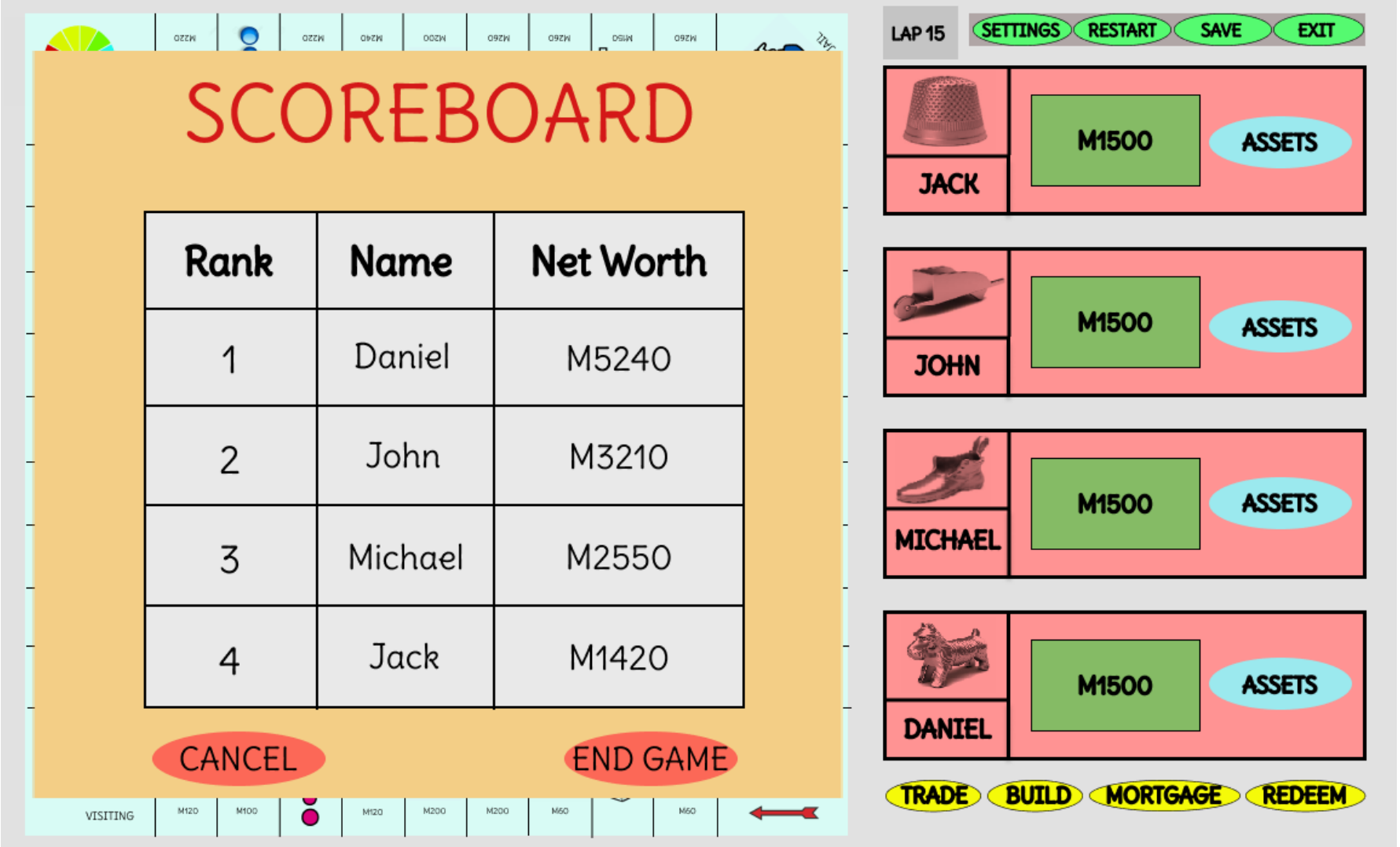

Figure 25. Gameplay end of the game screen.

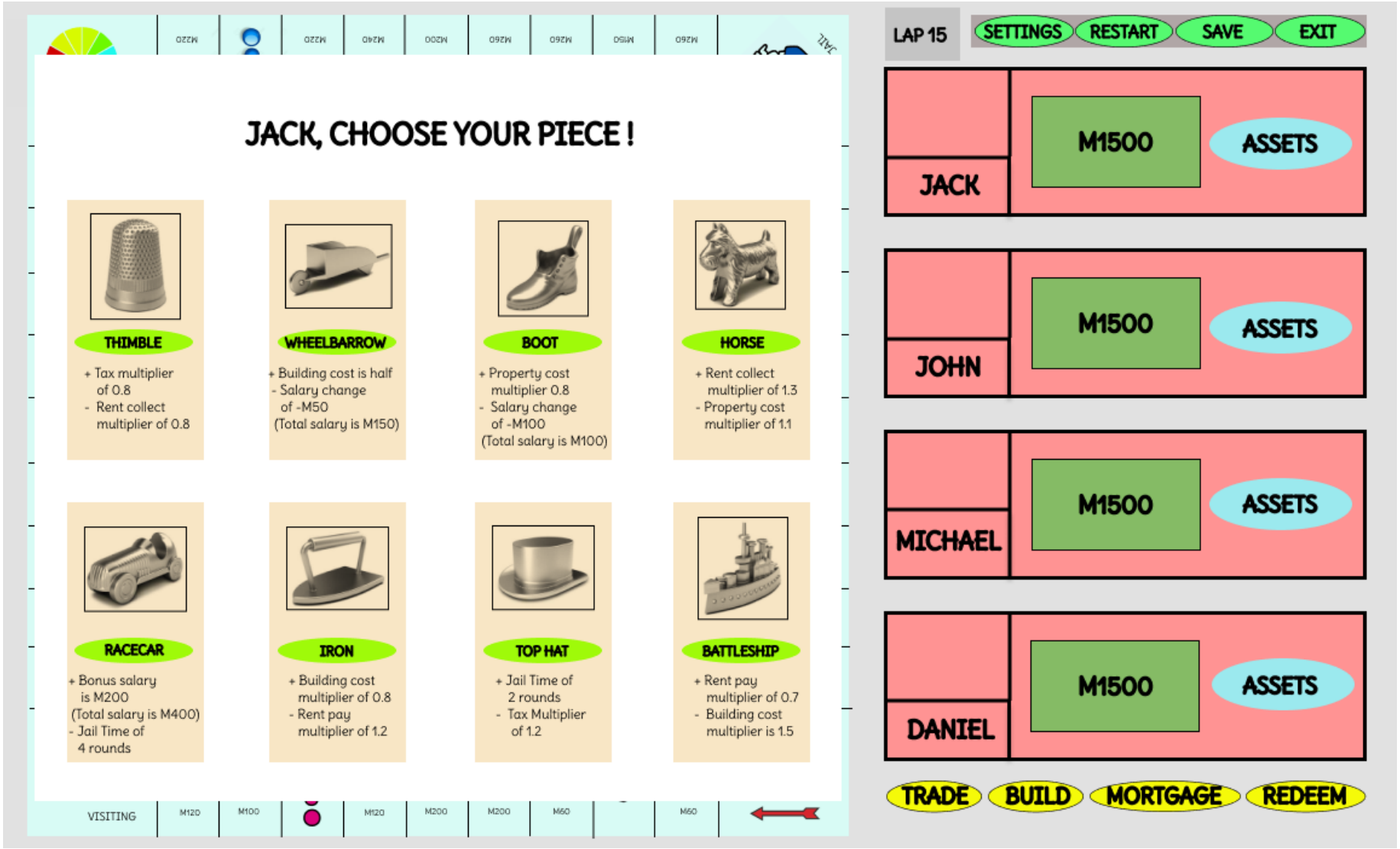

Figure 23. Gameplay choose avatar screen.

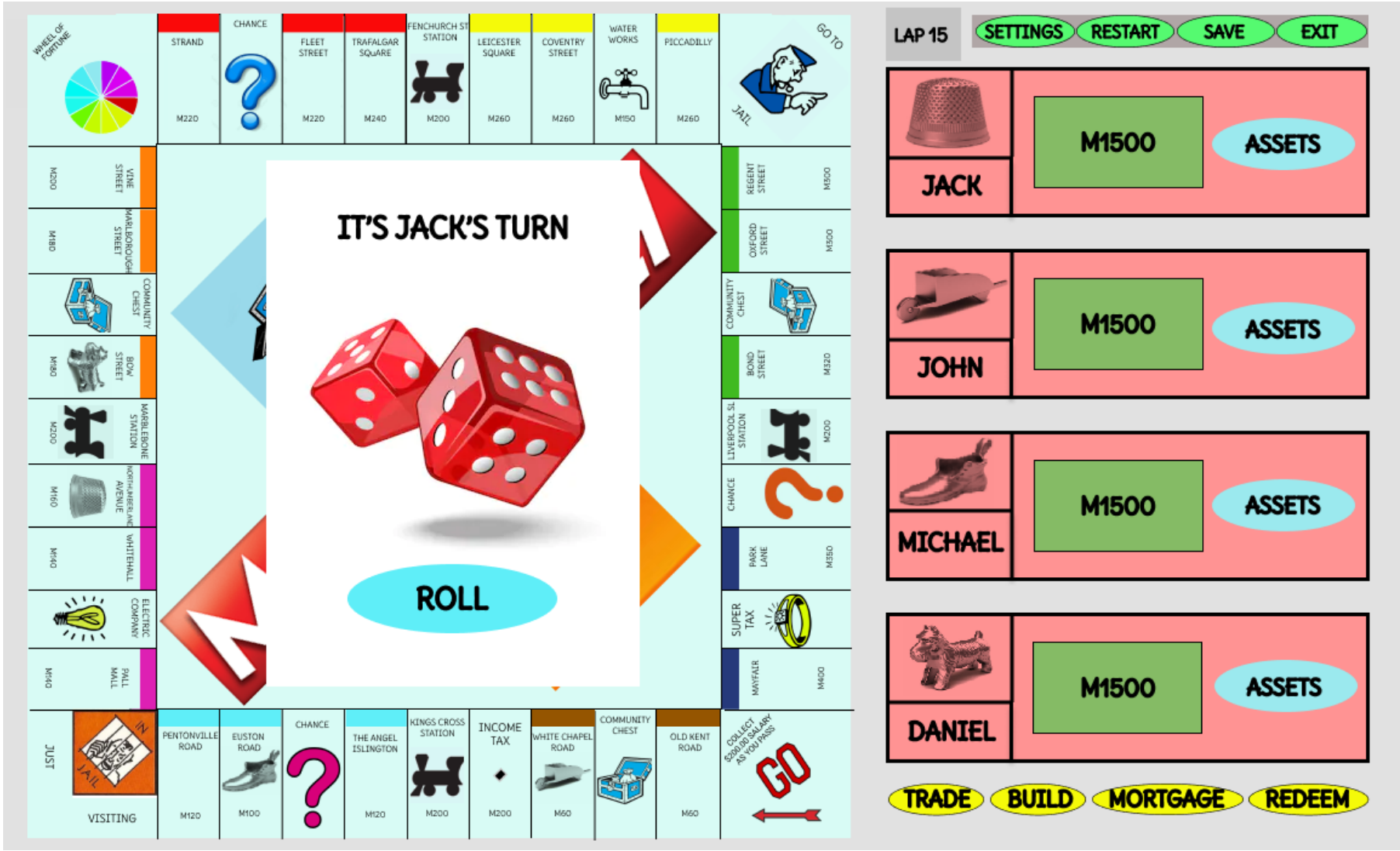

Figure 24. Gameplay roll dice screen.

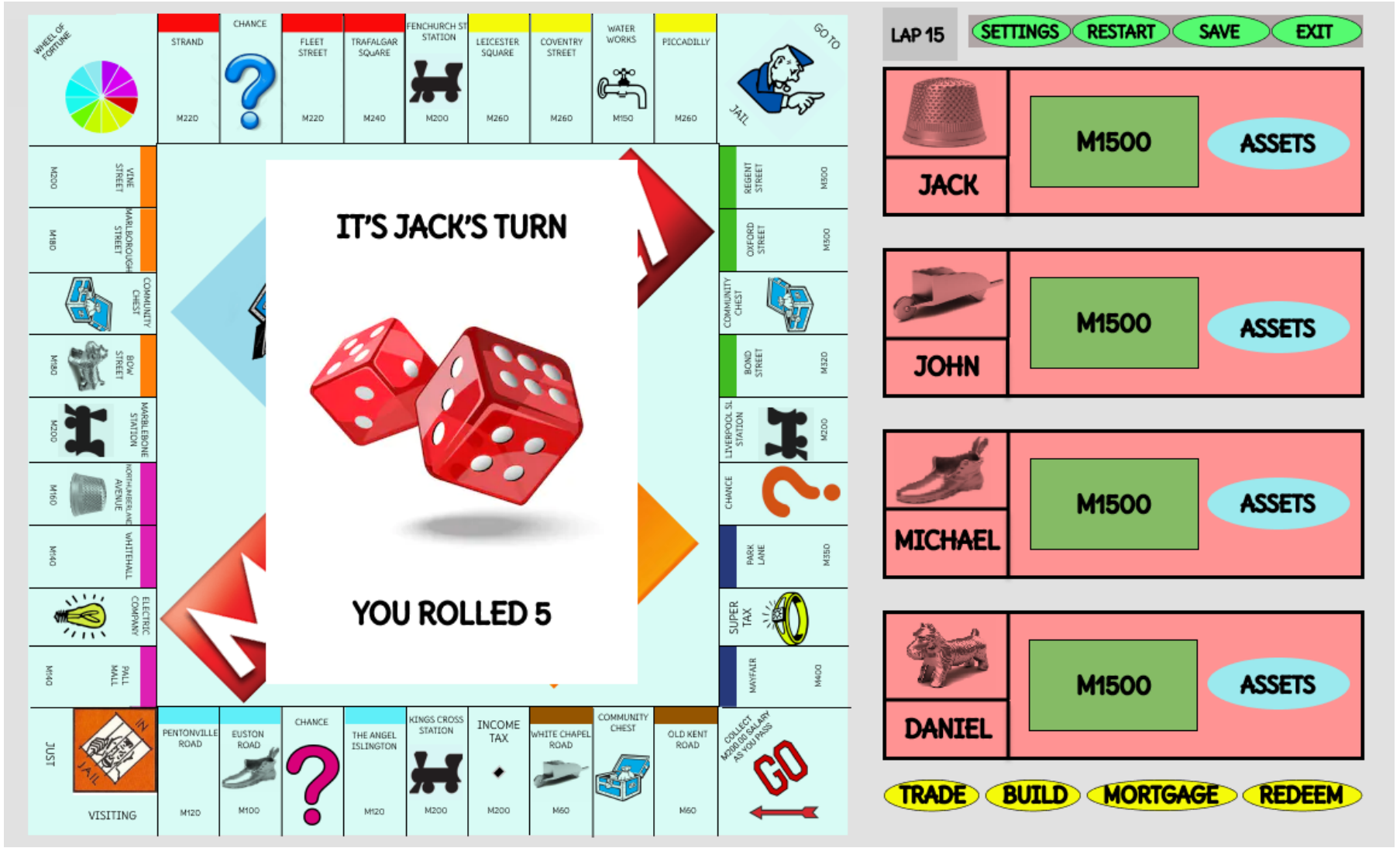

Figure 25. Gameplay roll dice and play screen

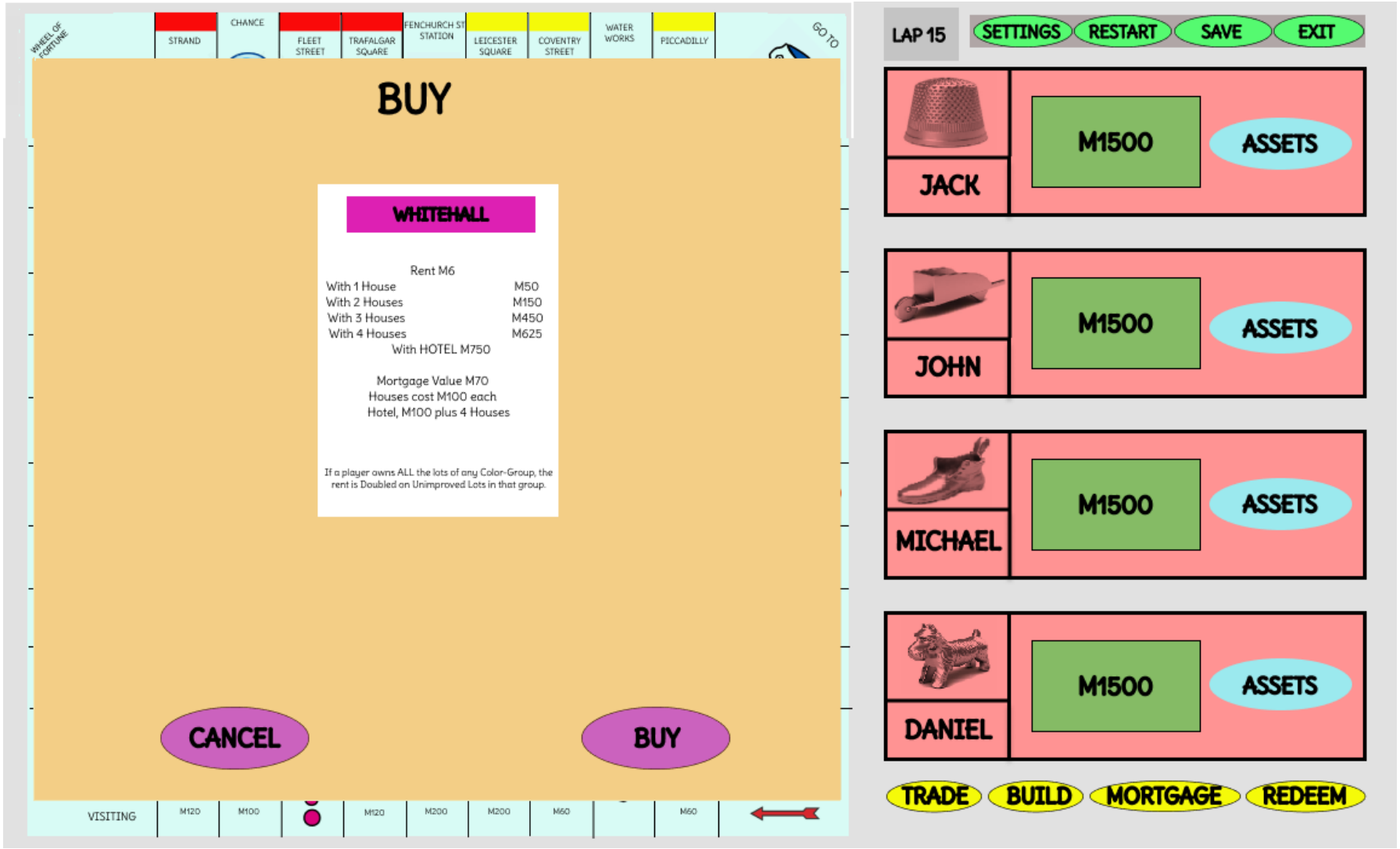

Figure 26. Gameplay buy screen.

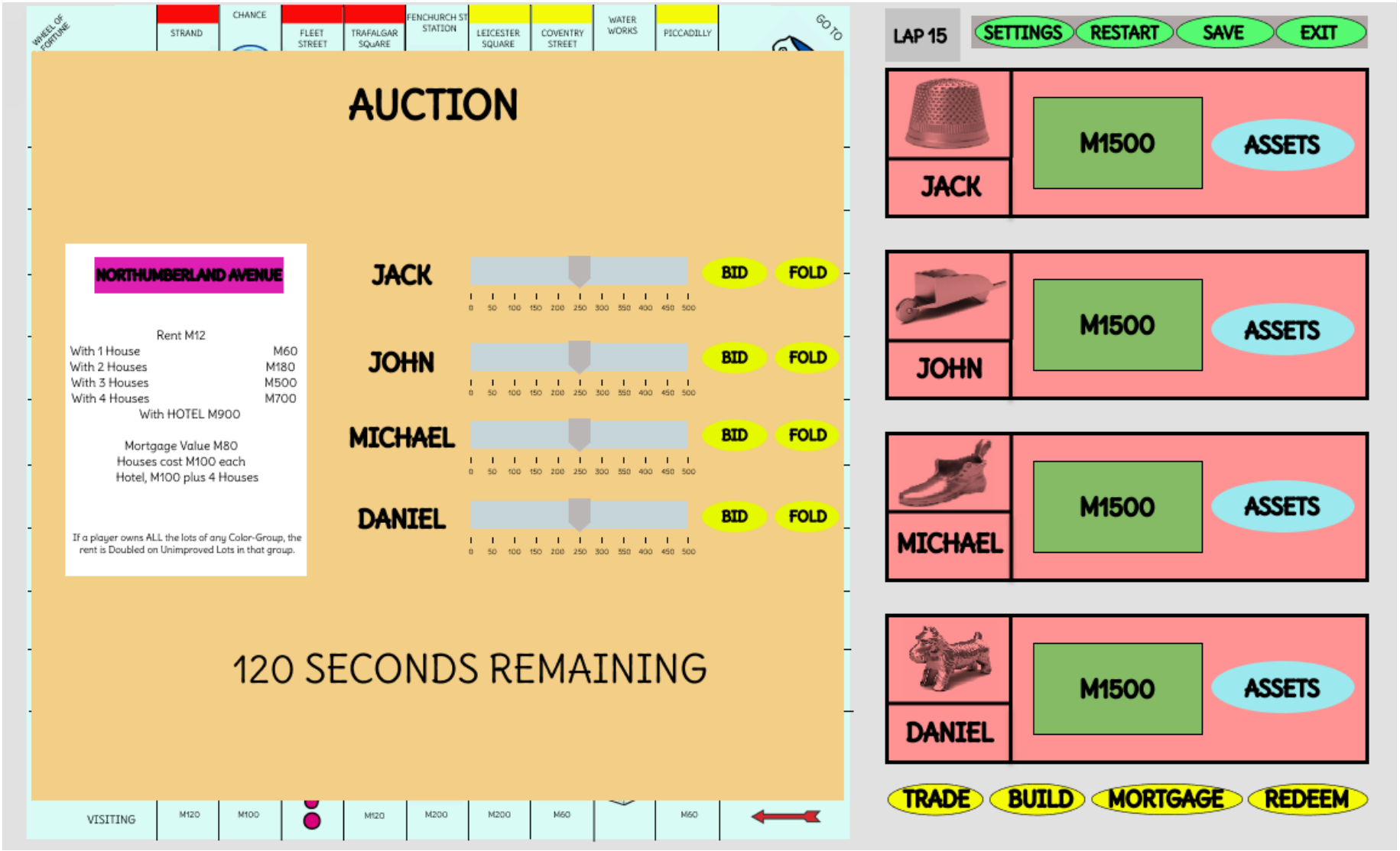

Figure 27. Gameplay auction screen

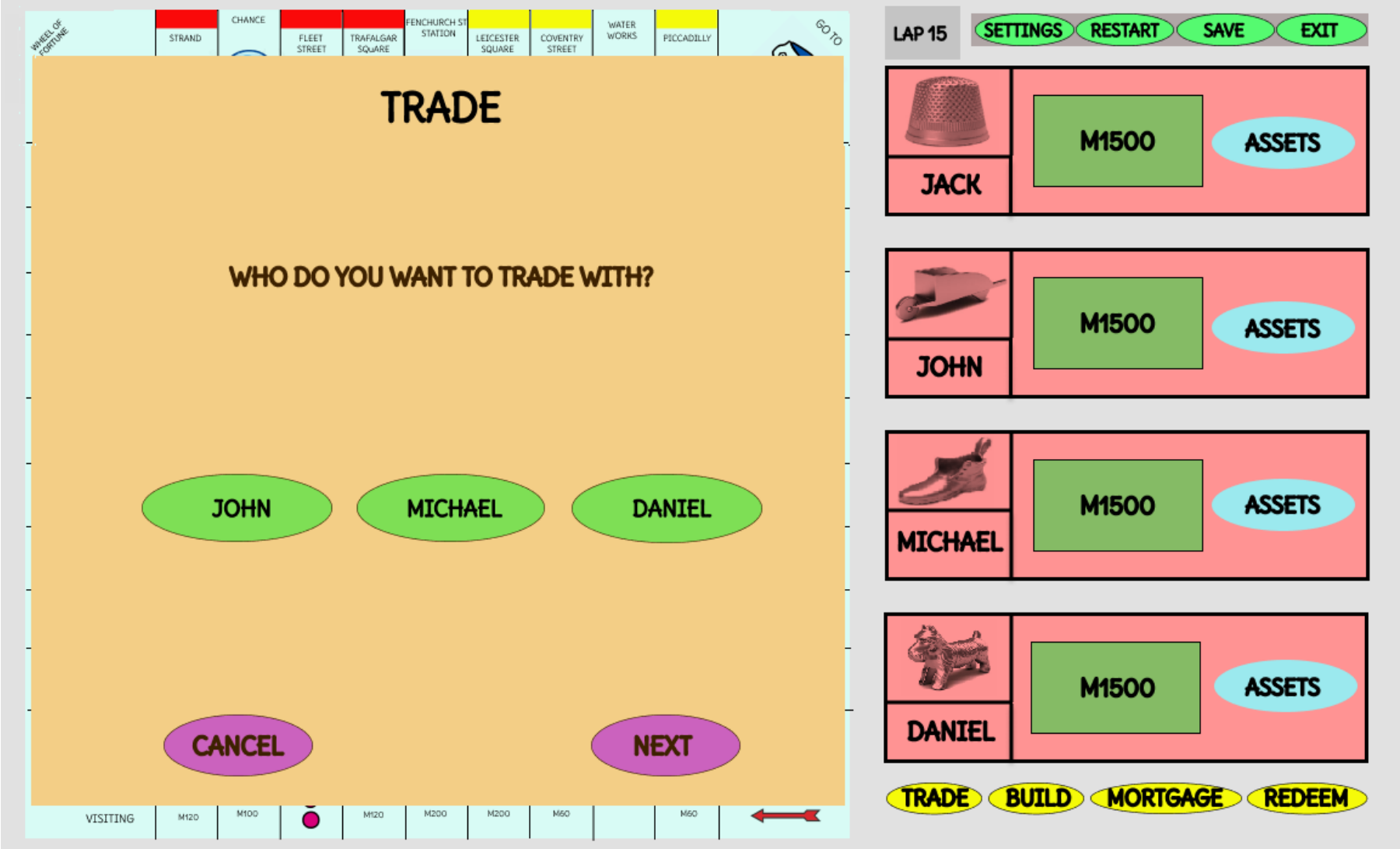

Figure 28. Gameplay trade player choosing screen.

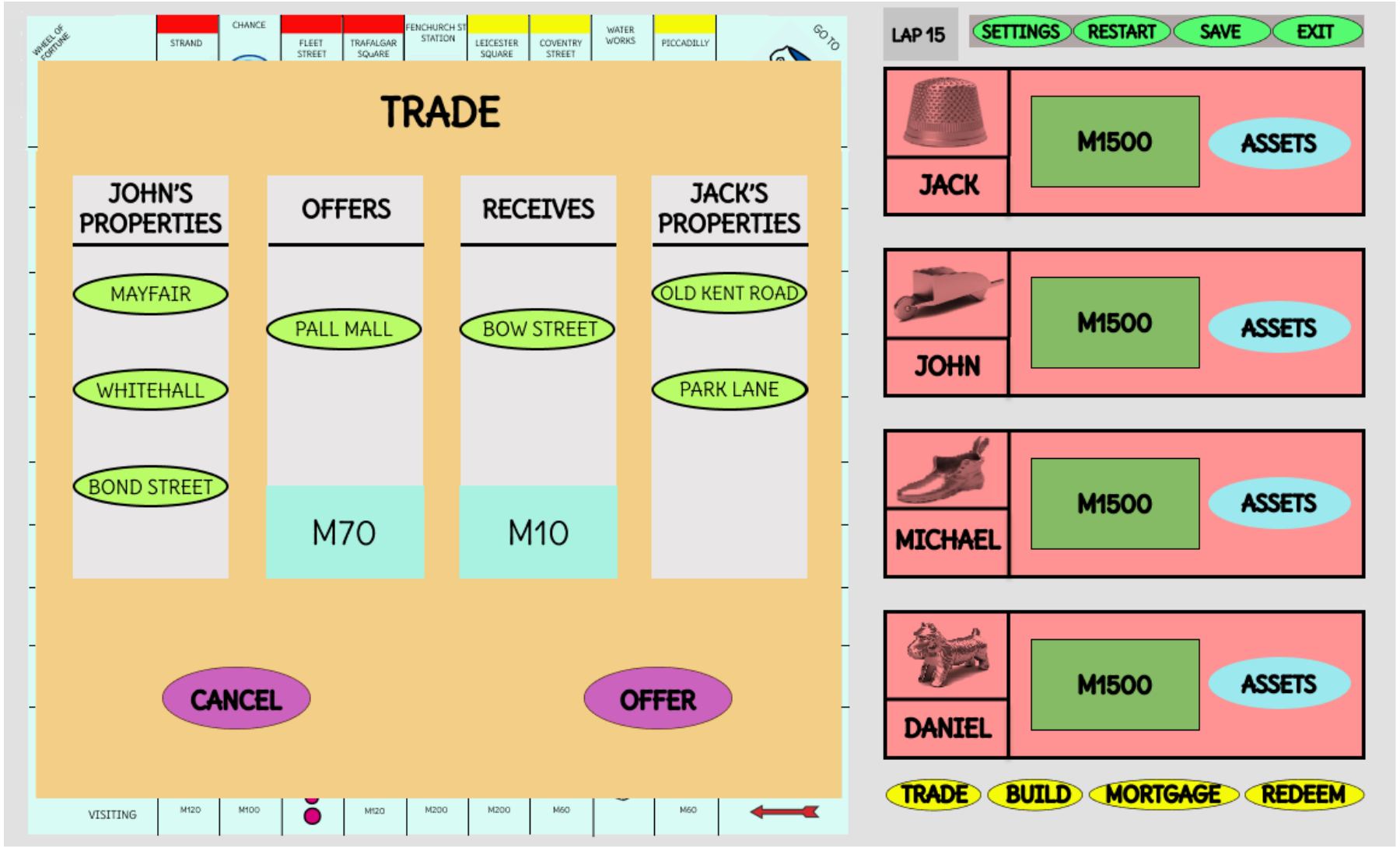

Figure 29 Gameplay trade offer screen.

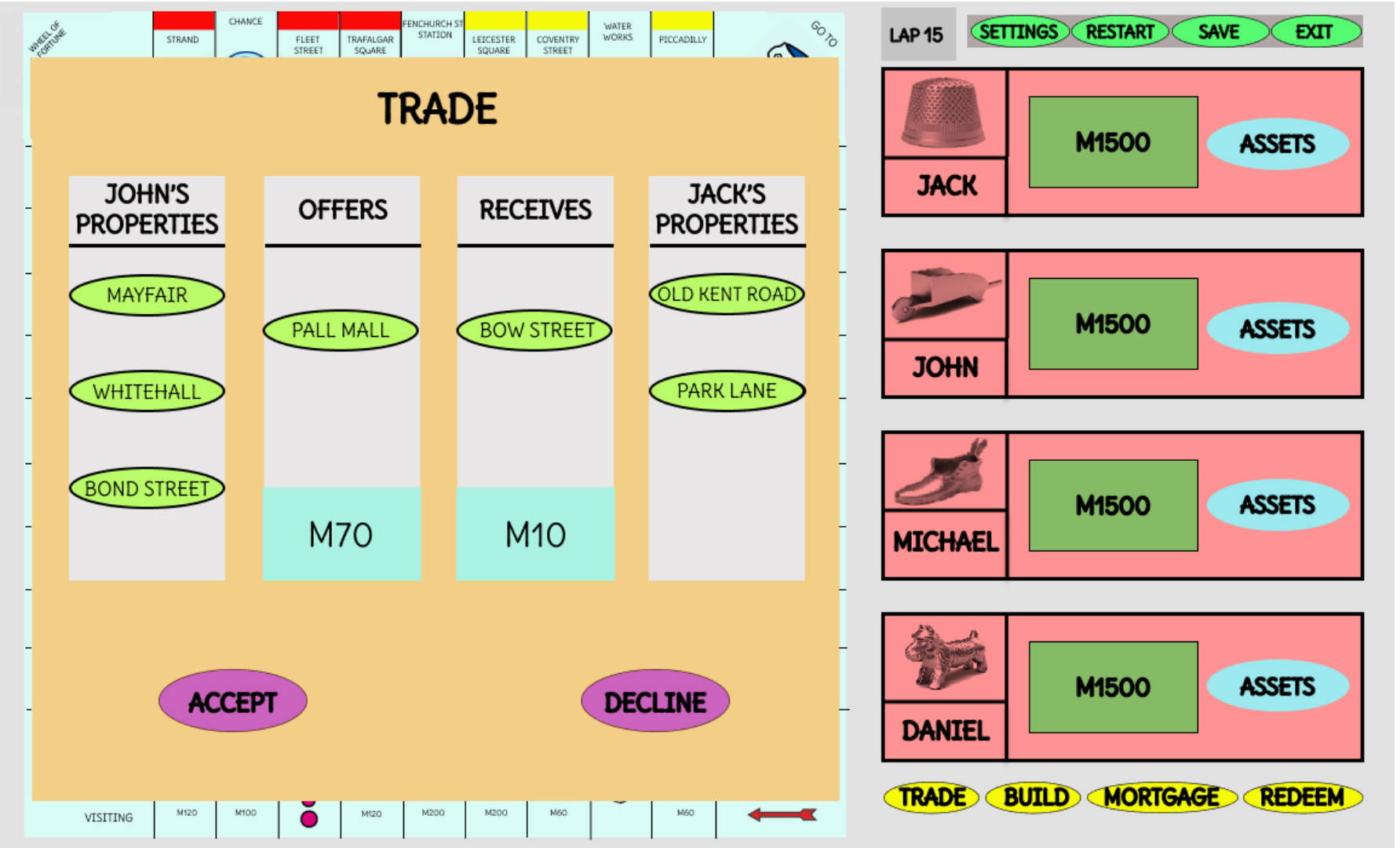

Figure 30. Gameplay trade approval screen.

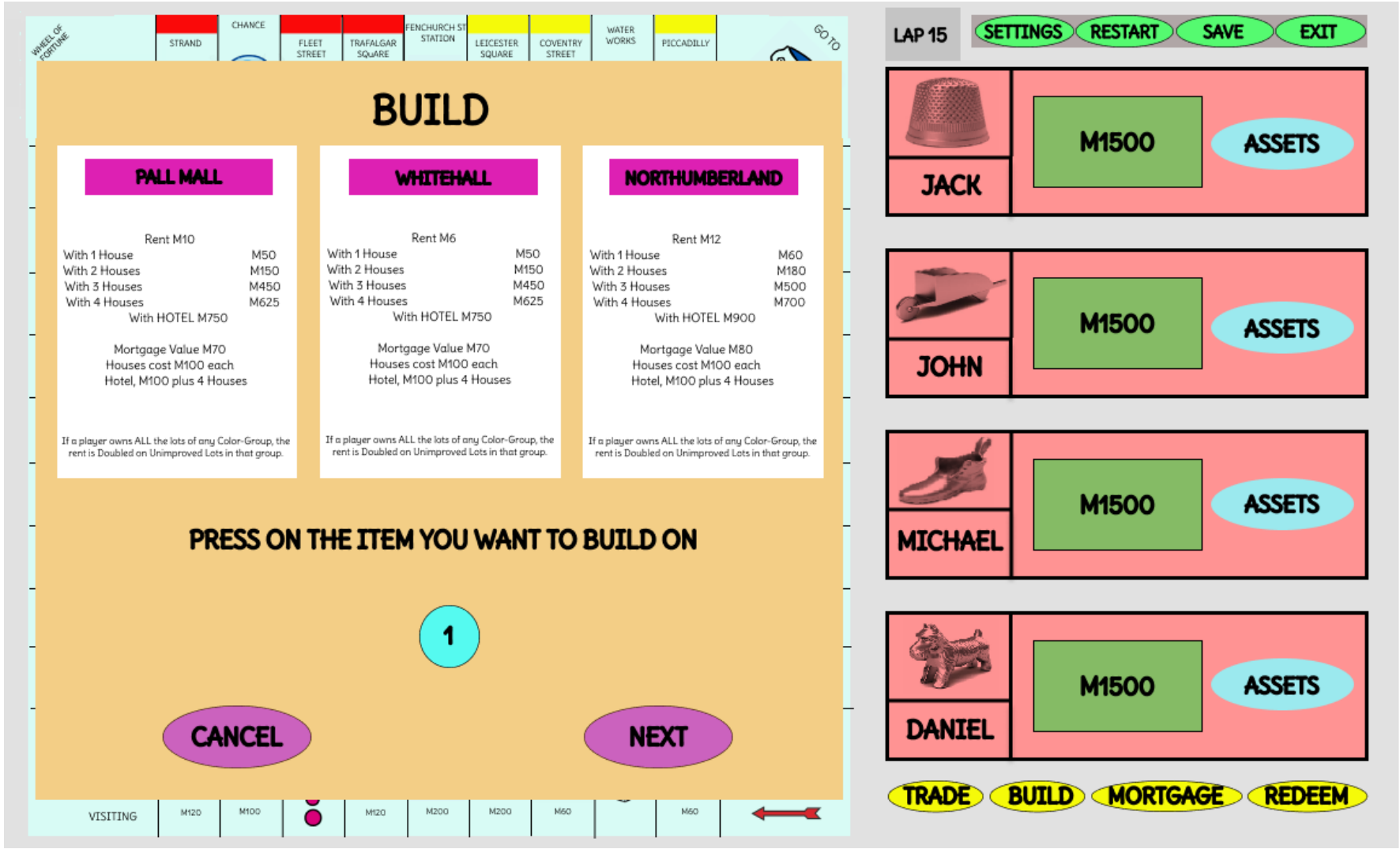

Figure 31. Gameplay build selection screen.

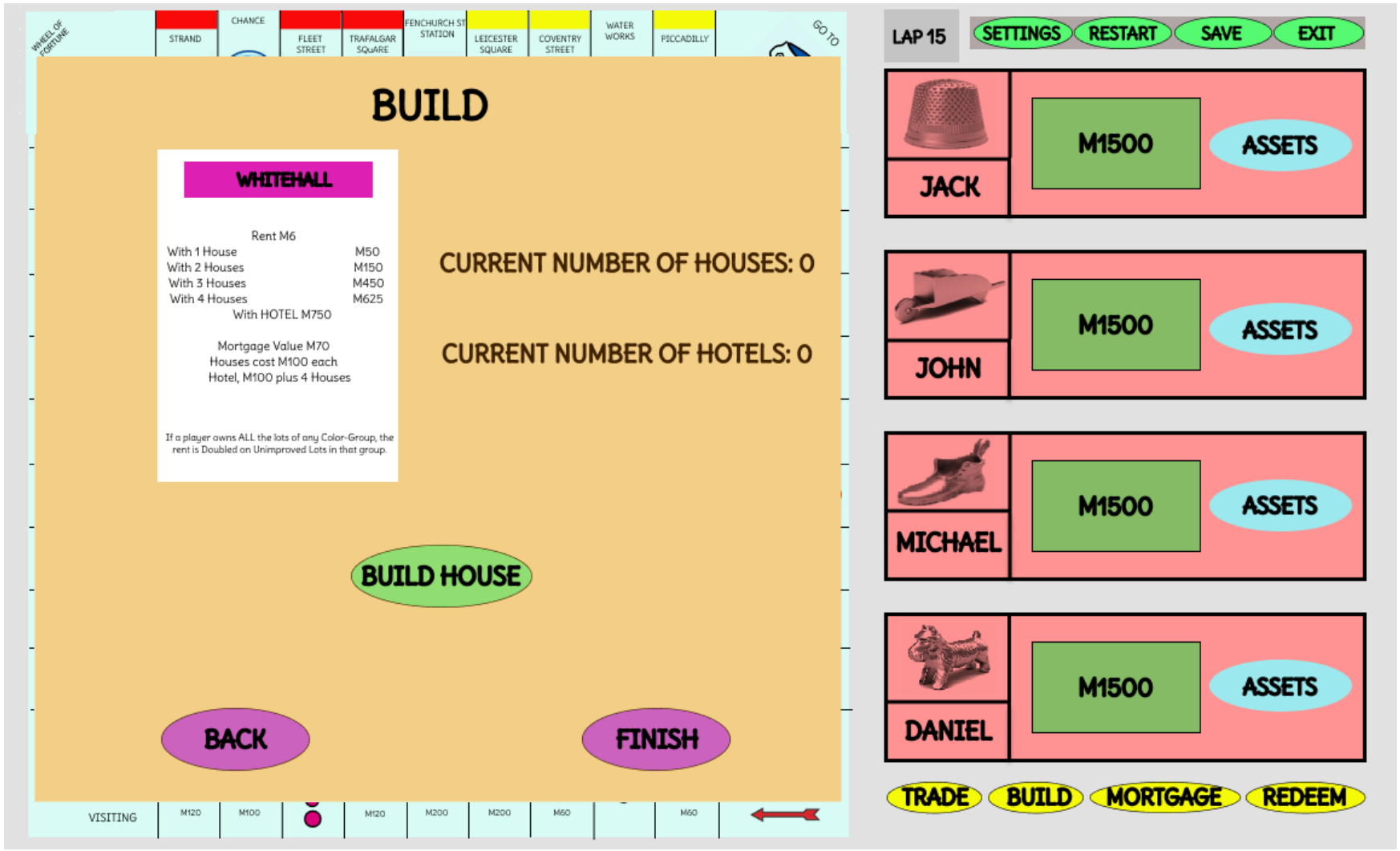

Figure 32. Gameplay build approval screen.

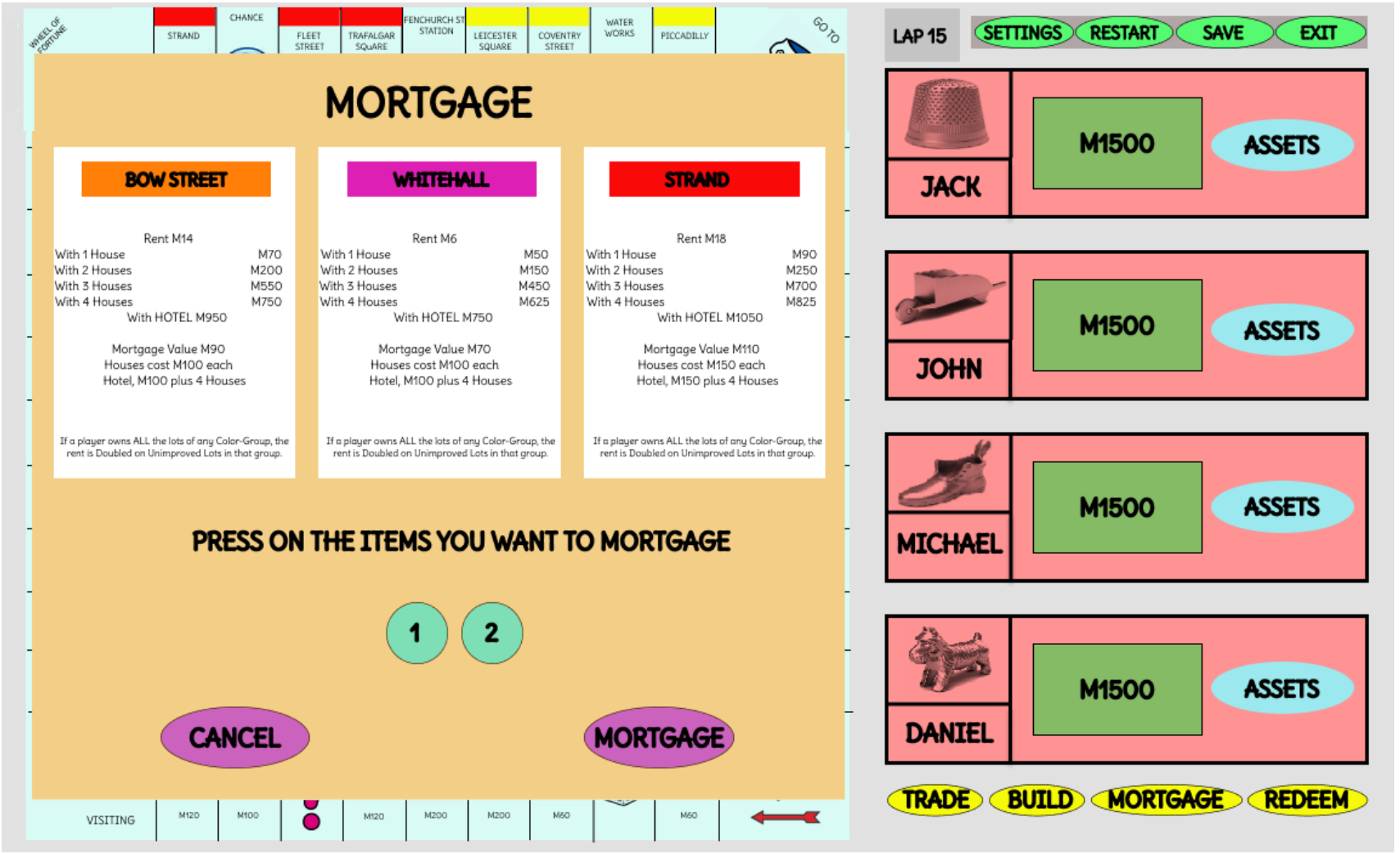

Figure 33. Gameplay mortgage screen.

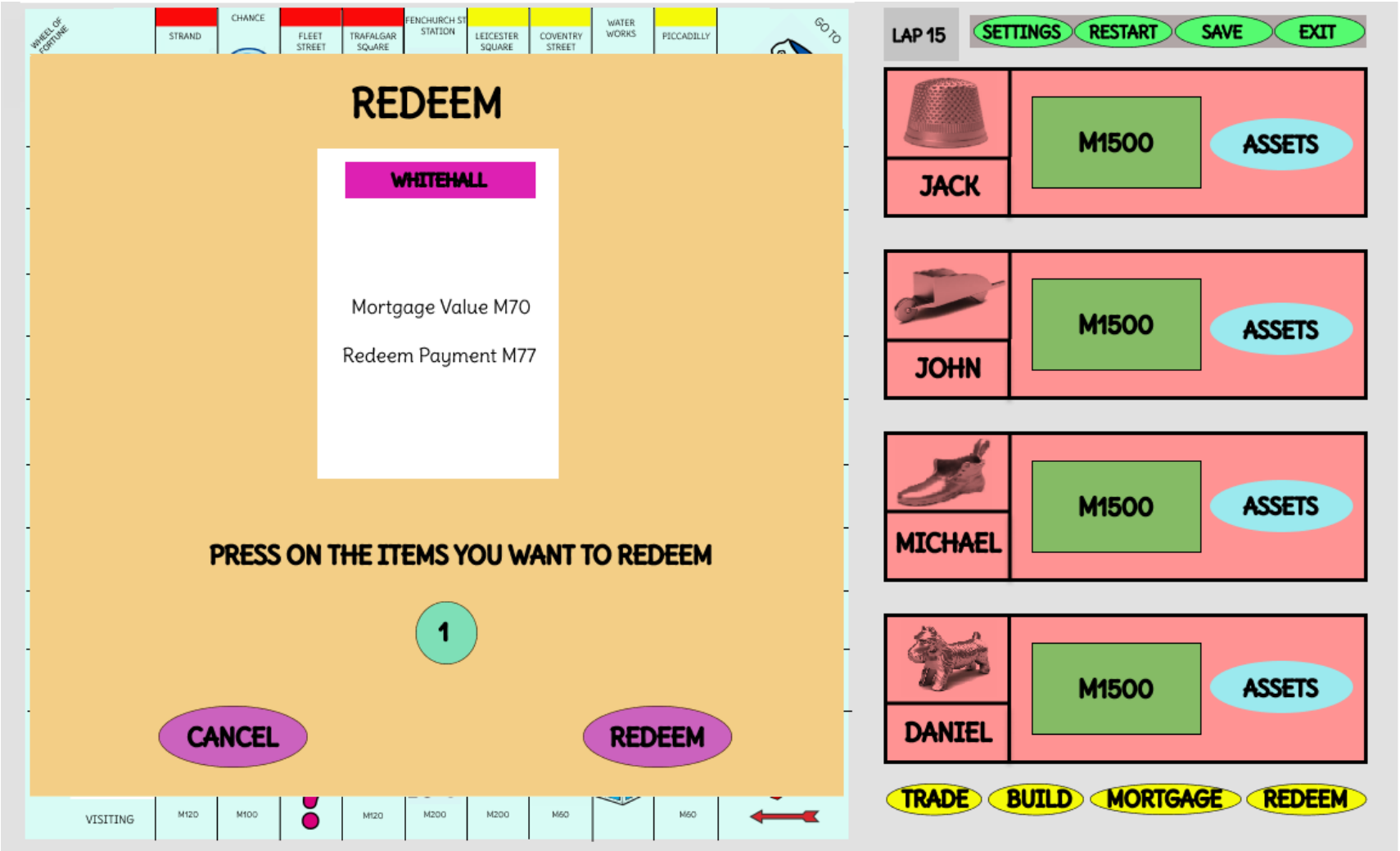

Figure 34. Gameplay redeem screen.

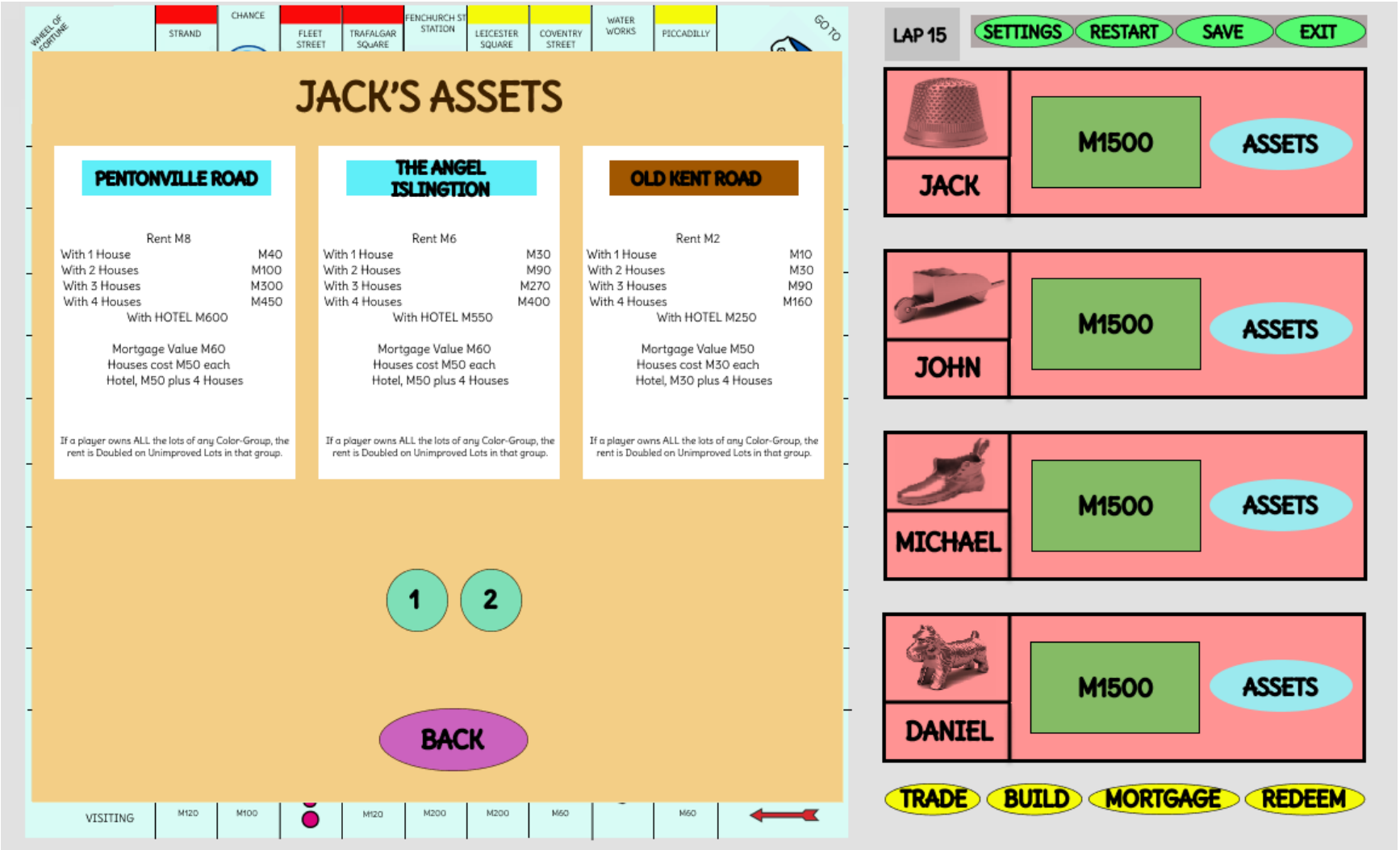

Figure 35. Gameplay assets screen.

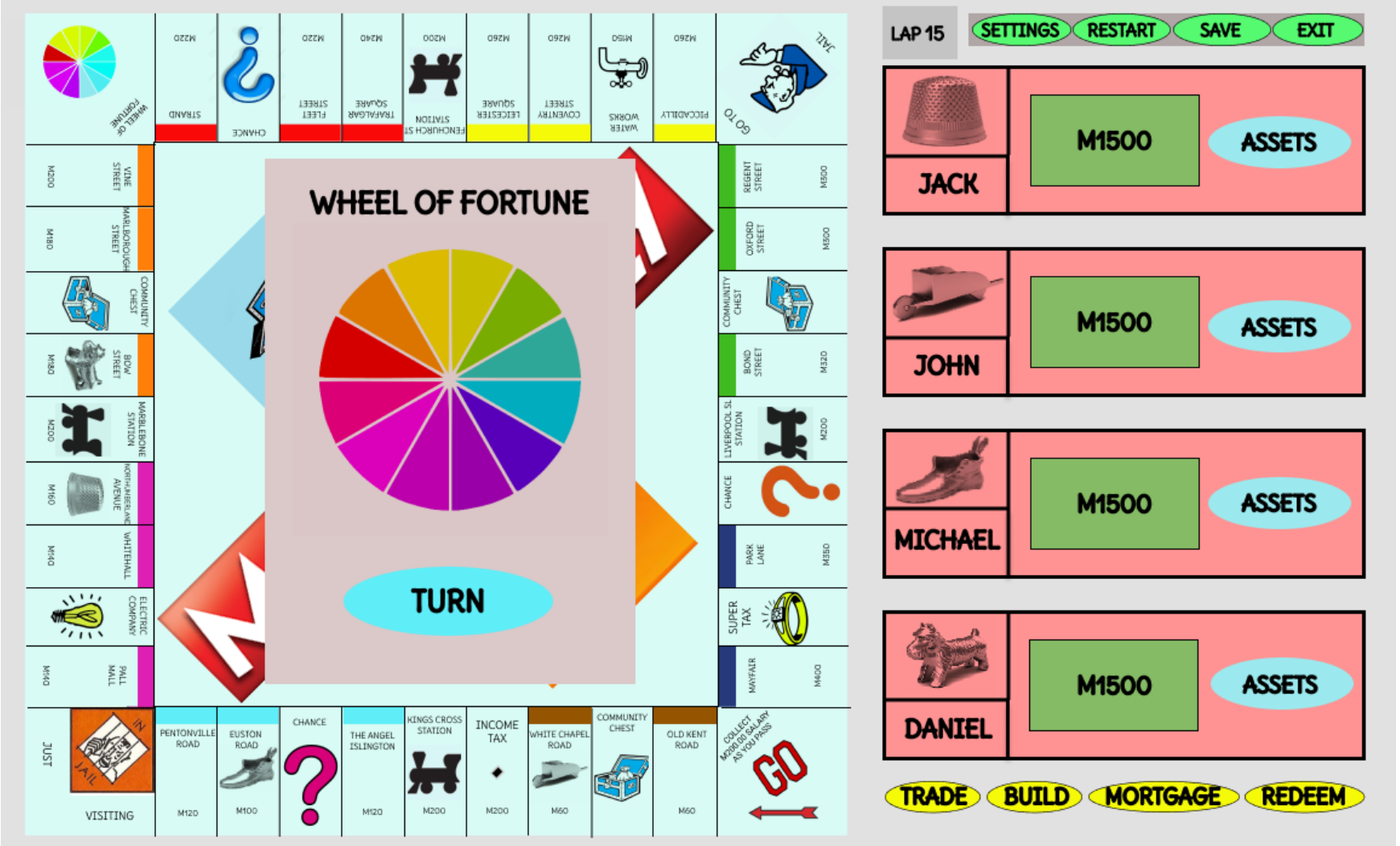

Figure 36. Gameplay wheel of fortune screen.

Figure 37. Monopoly game credits screen.

7 Bibliography

[1] *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.

[2] "Monopoly Rules." Hasbro, *www.hasbro.com/common/instruct/00009.pdf*. Accessed on September 29, 2020.

[3] Monopoly | Ubisoft (US), *www.ubisoft.com*, Accessed on September 29, 2020.

[4] IntelliJ IDEA Integrated Development Environment, *https://www.jetbrains.com/idea/*, Accessed on September 29, 2020.