CS-319 TERM PROJECT

Section 2

Group 2A

Monopoly

Final Report

Project Group Members:

1- Can Kırımca

2- Burak Yiğit Uslu

3- Cemre Biltekin

4- Can Kırşallıoba

5- Mustafa Yaşar

6- Emre Açıkgöz

Supervisor: Eray Tüzün

Table of Contents

1. Introduction

This section briefly discusses the final state of the project, including the progress and process of implementation and the changes to the design since the submission of iteration 2 reports under this section.

Under the section 2 “Lessons Learnt” we shared our reflections and discussions regarding this project’s components, including both the reports and the implementation. Solid, concrete matters about what we would have done differently if we were to start the implementation again are also discussed under this section.

Under section 3, “User’s Guide” a brief explanatory guide for the user for every GUI screen is included. This section is more focused towards explaining more complicated and possibly more unfamiliar features/actions for the users, namely the board building feature and the flow of the game once it is started.

Under section 4, “Build Instructions & System Requirements”, what system requirements are necessary to run the game, as well as building and starting the system is discussed.

Finally, under section 5, “Conclusion”, includes our final reflections on the project.

2. Implementation Process

We started the implementation once the skeleton of our design was finalized. Our repository was already ready in GitHub. Initially, the development progressed from 2 branches, one branch was dedicated to the development of the GUI, and other to the development of model and data subsystems. At the later stages, these branches were merged into an integration branch, where we continued the integration of the model and data classes to the UI one step at a time.

2.1 Important Note For Grading

In the Github repository, during the merging of different branches, due to inexperience and some technical difficulties, the integration of 2 branches were done by adding the codes of model classes to the gui classes manually as a new commit

and not by git-merging. As a result, for tracking of “who did what” in Github, especially with regards to model classes, the grader should check the history of the unstable-model branch as well, in addition to the master branch.

2.2 Communication

Throughout this implementation process, sometimes we worked individually and sometimes we worked in groups. For working in groups, we used a Discord server dedicated to our project that had numerous text channels as well as voice channels. In this Discord server, different tasks were divided into different text channels (for example, there was a UI text channel) and everything other than code was shared and let known to other players through these channels.

In the end, we managed to implement the core of the Monopoly Game, our extra features an aesthetic UI for what we have implemented. Most importantly, a maintainable and extendable Java code base.

2.3 Where We Are At The Implementation

At the end of the implementation process, we have a functioning game that one can play with other human players via local co-op or against AI Players. We implemented most of the basic functionalities of the game, as well as extra features we desired. However, due to time constraints, a small number of functionalities and features could not be implemented.

To briefly mention what the implementation is missing at its submitted stage, firstly, the auction system is not functionally fully implemented, therefore, not usable and as a result of this, when a player lands on a tile and does not buy it, the tile is left as it is, and an auction is not started. Secondly, by the original rules when a player should have gone bankrupt due to debt to another player, his/her properties needed to be transferred to the other player. This could not be implemented. Thirdly, when a double dice is thrown, the throwing player needs to play again, however, this could not be implemented as well. All of these features are missing due to time constraints and given enough time, we firmly believe that we could have implemented all of them.

Despite these, the game is aesthetically pleasing, reliably running and with the features implemented, could be played as a very fun game against friends or AI players.

2.4 Changes To The Original Design

Due to a variety of reasons regarding implementation, whether that be ease of implementation or unforeseen circumstances, various new attributes and methods were added to numerous classes such as boolean flags, getter and setter methods, or other auxiliary functions. These changes do not change the design.

One important change is to the graphical user interface, where various classes including view and controller classes were added. Because of our inexperience with JavaFX, we could have not foreseen the final state of the GUI classes before the implementation. Furthermore, because of the sheer volume of different classes, it would have not been possible for us to add them as well.

Another change is to the AI subsystem, where originally, extending strategies would all implement and override the playTurn method of the AIStrategy, which would be called by AIPlayer. We changed this in a way that now, AIPlayer calls its strategies' methods for specifically making decisions and executions of decisions regarding a particular tile, which are called when the AI player lands on a particular tile when playing its turn. This was done because a considerable part of the contents of the playTurn method of AIPlayer was common for the extending strategies as well, and to avoid writing/copying the same code over and over, it was implemented in this way.

3. Work Allocation

During the implementation, even though at different points in the game people or smaller groups were assigned particular tasks, for the most part, almost everyone worked in almost all of the subsystems in various ways, whether that be creating a structural skeleton code of a class, creating a base code from that skeleton code, thoroughly implementing these specific classes, performing code reviews for the said classes, integration and implementation of the communication of that class with the rest of the project, or simple bug fixing. Every team member has had a particular set

of classes that he/she worked on the most, but also everyone has taken roles in every subsystem as well. Even though everyone worked together, to mention who was “mainly” responsible for which parts of the report and implementation classes:

3.1 Emre Açıkgöz

I contributed to almost every part in all stages actively.

- **Analysis Report Iteration 1:** Application domain level class diagram, Sequence Diagrams. General decisions.
- **Design Report Iteration 1:** Solution level class diagram, Subsystem Decomposition diagram. General decisions.
- **Analysis Report Iteration 2:** Class diagram, Sequence Diagram**,** General decisions.
- **Design Report Iteration 2:** Solution level class diagram, Subsystem Decomposition diagram. General decisions. Design patterns, General Architecture.
- **Final Report:** Contributed collectively with all other team members
- **Implementation:** Managed the general collaboration and integration on the project for both the reports and the implementation. Cemre and I implemented the factories/builders in all subsystems. General bug fixing across all subsystems.

3.2 Burak Yiğit Uslu

- **Analysis Report Iteration 1:** Introduction & overview part, a part of sequence diagrams, state machine diagrams and their explanations, my share of the class diagram (⅙ of it), glossary, general editorial organization.
- **Design Report Iteration 1:** Subsystem decomposition diagram and its explanation, a part of subsystem descriptions, a part of section 3.4 class interfaces (explanation of class diagram and solution domain classes).
- **Analysis Report Iteration 2:** New state machine diagrams and their explanations, a part of the revamp of section 3.4 class diagrams.

- **Design Report Iteration 2:** Update of previously added parts, a part of the revamp of the section 3.4 class interfaces (explanation of class diagram and solution domain classes).
- **Final Report:** Contributed collectively with all other team members.
- **Implementation:** Classes of AbstractPlayer, PlayerToken, HumanPlayer, AIPlayer, GameStatistics, Dice, AIStrategy and extending BalancedAIStrategy, AdventurousAIStrategy, StingyAIStrategy, various enumerations and other minor structures in the Player & AI Subsystem, general bug fixing, with a focus on bug fixing on the AI and Player subsystems, creating necessary

3.3 Can Kırşallıoba

- **Analysis Report Iteration 1:** UI Mock-up screens, Navigational Path Diagram, ⅛ of the class diagram, explanations of the class diagrams
- **Design Report Iteration 1:** Hardware Software Mapping, Persistent Data Management, Access Control and Security, Boundary Conditions, explanation of the class diagram.
- **Analysis Report Iteration 2:** Same parts with Iteration 1
- **Design Report Iteration 2:** Same parts with Iteration 1
- **Final Report:** Contributed collectively with all other team members
- **Implementation:** (In collaboration with Can Kırımca) Each of the Tile classes and the ActionStrategy classes of those tiles. The classes in the property package In addition to these, bug fixing for the AI classes, implementation of two config files for the board.

3.4 Can Kırımca

- **Analysis Report Iteration 1:** UI Mock-up screens and navigational path, a part of the class diagram.
- **Design Report Iteration 1:** The sections “Hardware-Software Mapping”, “Persistent Data Management”, “Access Control and Security”, “Boundary Conditions”. Explanation of class diagrams.

- **Analysis Report Iteration 2:** Modification of the sequence diagrams, and the same parts with Analysis Report Iteration 1.
- **Design Report Iteration 2:** The same parts with Design Report Iteration 1.
- **Final Report:** Contributed collectively with all other team members.
- **Implementation:** (In collaboration with Can Kırşallıoba) Each of the Tile classes and the ActionStrategy classes of those tiles. The classes in the property package. General bug-fixing (Mostly for the Tile classes). Preparation of the default configuration file and a few configuration files for testing and bug fixing.

3.5 Mustafa Yaşar

- **Analysis Report Iteration 1:** Use Case Diagram & its explanations, Overview, Activity Diagram, ⅛ of the Class Diagram
- **Design Report Iteration 1:** Introduction, some part of the Low-level design, explanations.
- **Analysis Report Iteration 2:** Activity diagram corrected according to the feedback.
- **Design Report Iteration 2:** Same parts made in Design Reports iteration 1.
- **Final Report:** Contributed collectively with all other team members.
- **Implementation:** All of the graphical user interface that can be seen in the game

3.6 Cemre Biltekin

- **Analysis Report Iteration 1:** Use Case Diagram & writing/editing all of its explanations, Activity Diagram & its explanation, ⅛ of the Class Diagram, Non-Functional Requirements
- **Design Report Iteration 1:** Introduction & Design Goals, Subsystem Architecture explanation (why we chose MVC and 3-tier) & layers' explanations, Object Design Trade-offs, Packages, Class Interface explanations

- **Analysis Report Iteration 2:** Same parts made in Analysis Report Iteration 1, making of Board Building Activity Diagram & Inside Game Activity Diagram, Improvements Summary
- **Design Report Iteration 2:** Same parts made in Design Report Iteration 1 but in more detail, Improvements Summary
- **Final Report:** Contributed collectively with all other team members.
- **Implementation:** LoadGameController, NewGameController and BoardBuilderController, BoardBuilder, ConfigHandler, FileManager, SerializationHandler, TileFactory, BoardFactory, some contributions in other factory classes and management of saves and loads in model and controllers & board building feature. Bug fixes.

4.Build Instructions & System Requirements

This game runs on Java. In order to be able to run the game, one needs the latest version of Java 8. In order to compile the game, one needs to have the latest version of JDK for Java 8. In particular, this game was built on and tested on version JRE 8u201 and JDK 8u261. In order to play the game pleasantly, one needs to have a screen that supports at least 1920x1080 resolution.

The game is built for the

1. Install Windows 10 to your computer.
2. Install jre1.8.0_271 and JDK 8u261 or newer versions.
3. Add bin directory of the jdk to path variables.
4. Install Intellij IDEA (version 2020.3 or newer versions) for Windows.
5. Download the source code of the project. Source code includes the dependencies.
6. Extract the source code from zip file
7. Import the source code as an Intellij project. You should import the **folder** named CS319_Group2A_Project_Monopoly-master. From now on you have two choices to run the game: Using IDE to run the game, or building a Jar file (Be careful you must import the directory which contains the src and resources file).

The file structure of the directory that needs to be imported as follows:

CS319_Group2A_Project_Monopoly-master\
src\
resources\
board_default_template.json
board_fast_gameplay.json
board_template.json
...

a. If you have opened a different project previously, you need to close the current project (File -> Close Project)
b. You can import the project from the *Welcome Screen*

8. Using IDE to run the game:
    a. Add Main class to configuration
        i. Press *CTRL + SHIFT + A*, Type *Edit Configurations* and then press Enter.
        ii. You should see the *Run/Debug Configurations* window now.
        iii. Press *Alt + Insert.*
        iv. Select "*Application*" selection from the pop-up window.
        v. Specify the Main class by clicking the text box for it.
        vi. Press *Shift + Enter.* On the pop-up window, find the Main class by typing "Main" into the search textbox. After selecting it, click OK.
        vii. On the *Run/Debug Configurations* window, click OK.
        viii. Press *Shift + F10* to compile and run the game.
9. Building A Jar File:
    a. From the top menu, click the following labels and follow these screen/menu names in the specified order: File -> Project Structure -> Project Settings -> Artifacts -> Click plus sign -> Jar -> From modules with dependencies… (Before this step, you might need to delete the previously defined configuration by pressing the minus sign after selecting the previously created configuration.)
    b. Specify the Main class. Then, click OK.
    c. Click OK on the Project Structure window.

d. From the top menu, click the following labels and follow these screen/menu names in the specified order: Build -> Build Artifact -> Build.
e. Find the Jar file in the directory: $PROJECT_ROOT\out\artifacts\CS319_Group2A_Project_Monopoly_jar
f. Copy the following config files, which can be found in the top of the project root, to Jar file directory: board_default_template.json, board_fast_gameplay.json, board_template.json (depending on your configuration, the extensions of the files may not be visible)
g. $PROJECT_ROOT\out\artifacts\CS319_Group2A_Project_Monopoly_jar/CS319_Group2A_Project_Monopoly.jar file

5. Lessons Learnt

Throughout this course, we have learnt Software Engineering principles, how to function in a team and develop a project together, respond to changes; and we implemented this project using this knowledge. In the course of learning, we have learnt about analysis, design, and implementation processes by drawing UML diagrams, applying design patterns, making design and requirement decisions and implementing the game in collaboration.

In the analysis step, first, it was challenging to brainstorm about all use cases and corner cases to construct accurate and explanatory diagrams. Since it was the first step, we had just been beginning to grasp Software Engineering principles. Although the game or the product itself is a game that we have played many times, it was surprising to us how many aspects we had to consider to come up with the analysis. We have learnt that, as much as we may be familiar with the product that is wanted from us by the client, we have to contemplate hard to cleanly begin its implementation that shows these Software Engineering principles. We used to begin head-first with the implementation before taking this course, but we have realized that the analysis and design steps are equally and maybe more important than the implementation itself since an adequate analysis and design mean a better

functioning implementation that is open to extensions. Since it is a board game (physical), we had to imagine a digital version of it and progress in our analysis according to it by adding extra features. It was fun to come up with new additions to the game like bots, new tiles, and board customization features to make the digital version more exciting. Integrating these new features in our project was also challenging. Thus, we have gained experience about how to approach a client's expectations of a digital product and how to come up and present our analysis coherently. We had to elaborate enough to refer to all expectations, but at the same time we had to bear in mind not to be too technical (remain on the level of the client).

Before the design process, we learnt about design patterns, system architectures and design goals in the course. In the design process, we have learnt how to assess design goals and tradeoffs for the project, decide which design patterns and architectural decisions are suitable for it, and construct an object model that carefully and accurate displays these decisions. The most challenging part was to choose the right design patterns and construct the object model, since we had to specify relationships, each attribute and method according to it. We went through all use cases through the object model to check missing class/attribute/operation and add them; yet after the first iteration, we realized that we might need other components to serve features of the game. We learnt that a good analysis serves as a base for a good design.

After the design process and when we began implementing, we realized that design and analysis together serves as a guide together for the implementation. Thanks to design, we were able to implement the game and its features quicker than we had anticipated since we had limited time; and thanks to analysis, we knew what to test, usually meaning corner and use cases, in bug fix sessions. However, we also acknowledged that we had to deviate from the design time to time in terms of method and attribute usage to complete the implementation. We were glad that we mostly stuck to our initial design; but we also encountered surprise elements and changes which forced us to make alterations in code, too. In these times, we understood why we had to design our project regarding probability of changes and extension; we could make these alterations, additions, removals smoothly without changing much code thanks to our design choices. Moreover, we also learnt that

there are many bumps in the road of implementation, and working as a team, using our strengths and weaknesses, has helped us overcome them.

The allocated time for the implementation is quite short, therefore, it requires really hard work to create a properly functioning and good-looking implementation. It would be better to start the implementation part earlier; we could have tested more corner cases, fixed more bugs, fully completed our implementation, and maybe worked on response time of the game. Overall, we are content with this experience.

6. User's Guide

6.1 Menu Navigation Guide

The main menu consists of 6 options: Start New Game, Load Game, Board Builder, Credits, How To Play and Exit Game.

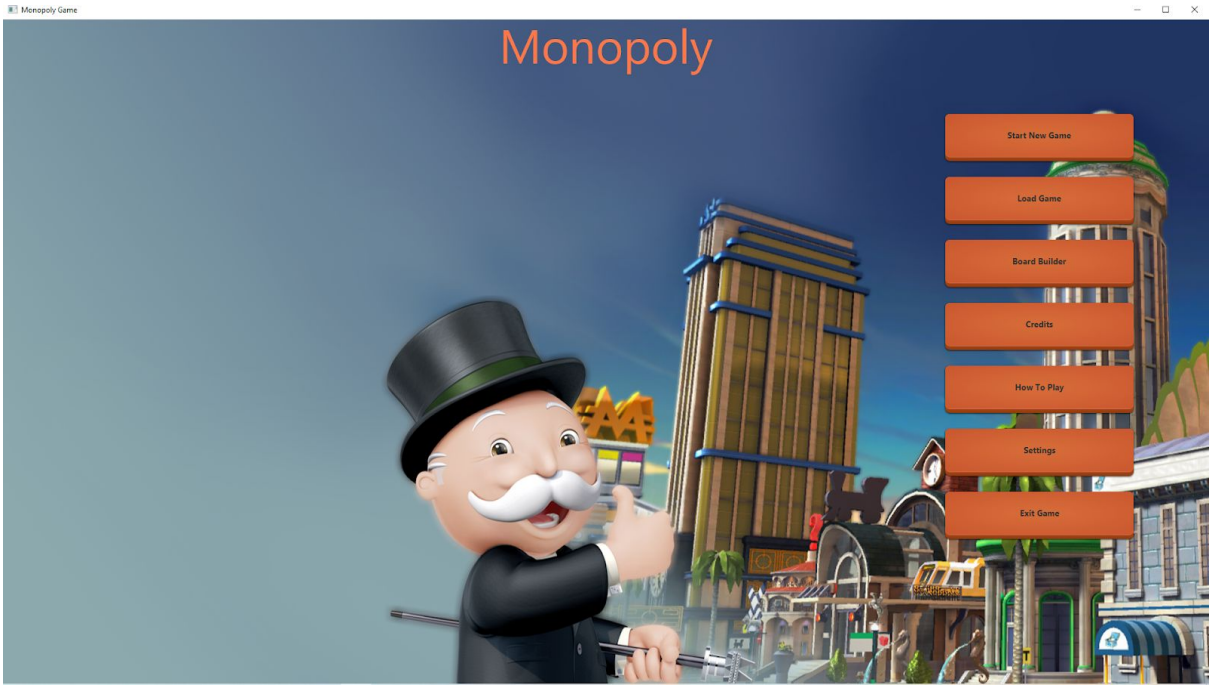

Figure 1. Main menu screen

6.1.1 Starting a New Game

The user can start a new game by clicking the "Start New Game" button. Before starting the game, the user can specify the features of the new game in the "New Game" screen. After designating the features of the new game, the user can start the game by clicking the "Start Game" button.

Figure 2. New game screen

6.1.2 Loading a saved game

The user can load a previously saved game by clicking on the Load Game button. The user can then select a save file from the list of files, then continue playing the game.

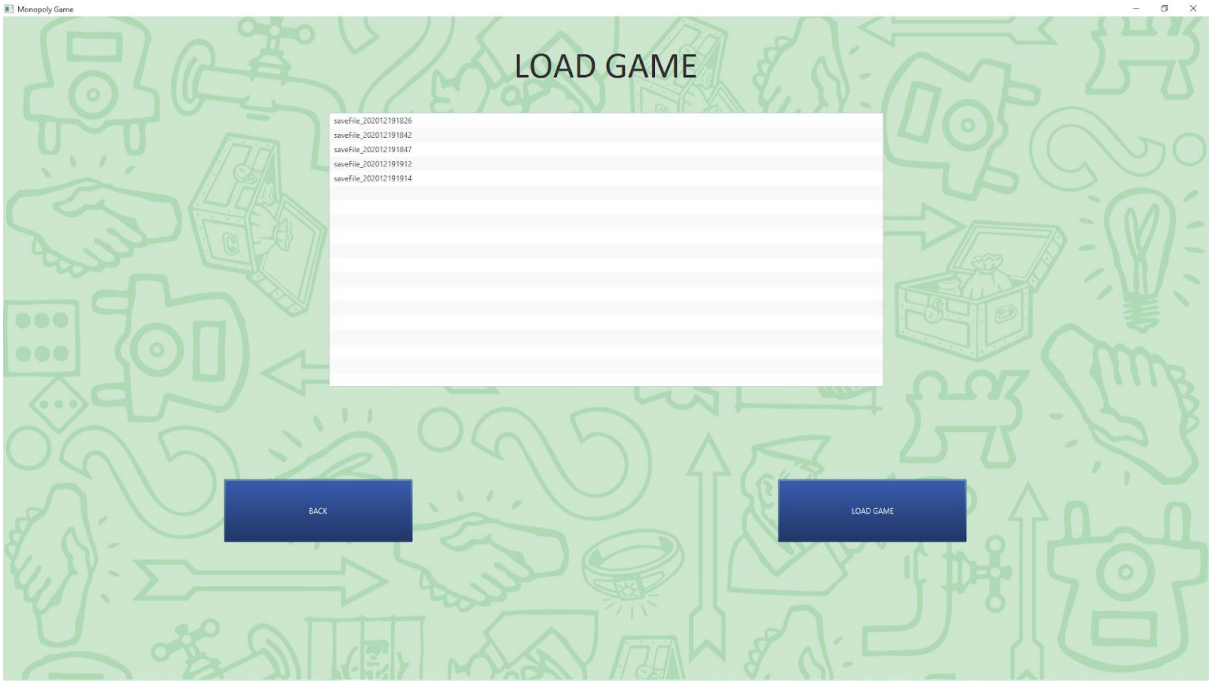

Figure 3. Load game screen

6.1.3 Building a Board

The users are able to customize their boards by entering the detailed information about the properties one by one. These customizations include the rent, name and the price of the property. It is crucial that the users enter all the required information regarding the properties because if they do not, it can lead to some undesired consequences in the gameplay. The boards can be saved for further usage in other sessions.

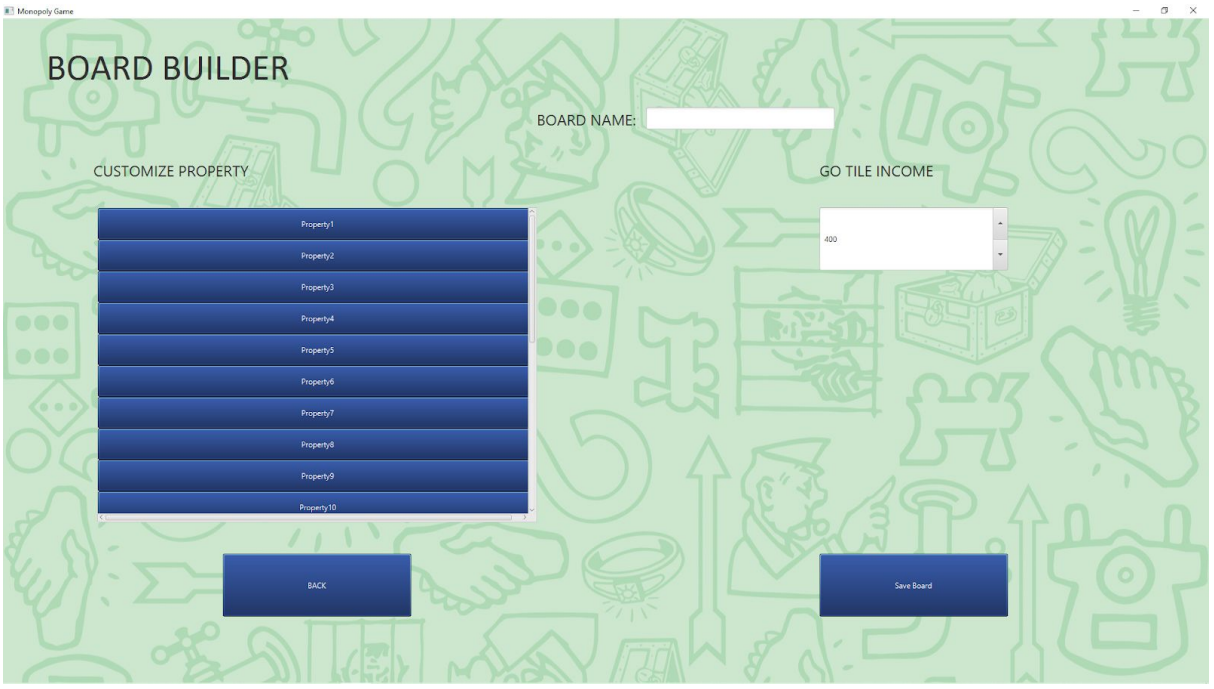

Figure 4. Board Builder screen

6.1.4 How to Play

The user can learn about the rules and the gameplay of Monopoly on this screen.

Figure 5. How To Play screen

6.1.5 Credits

The user can learn about the credits of the project on this screen.

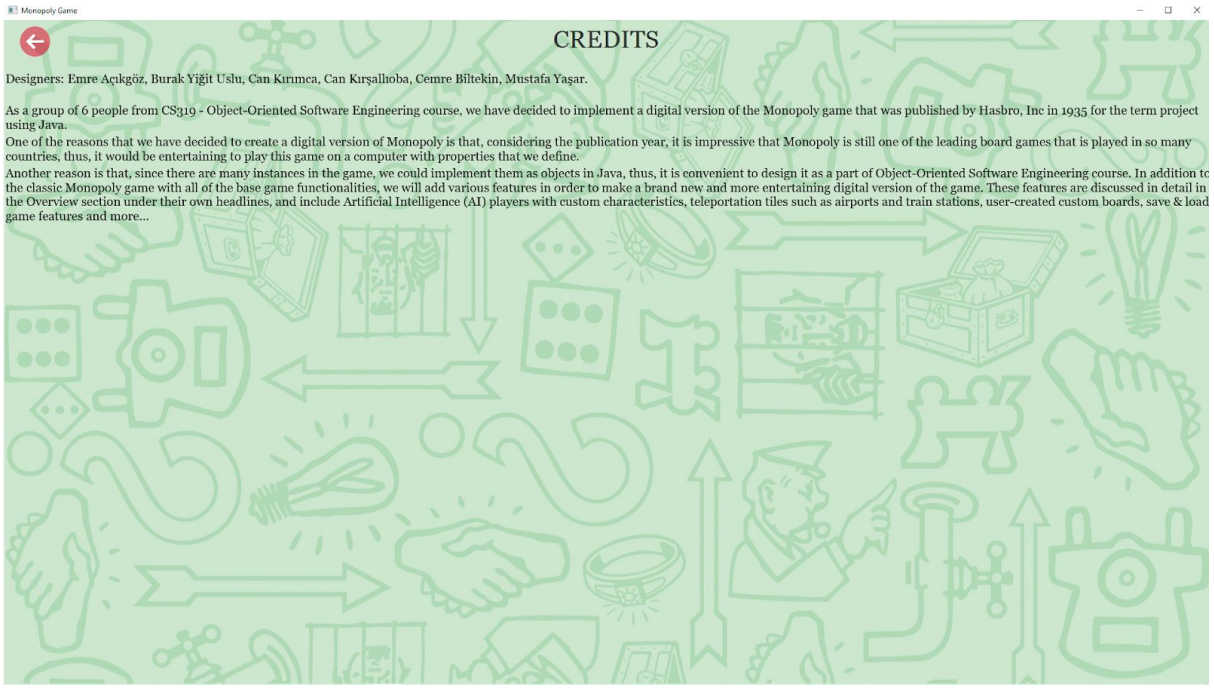

Figure 6. Credits screen

6.1.6 Exit Game

The user can quit the game by clicking the “Exit Game” button.

6.2 In Game Guide

6.2.1 Turn Sequence

The gameplay proceeds sequentially with player turns. Each player can perform their possible actions in their turn. After a player clicks the “End Turn” button, the game continues with the next player’s turn.

6.2.2 Moving Tokens

After rolling dice in a turn, the system automatically moves the player’s token according to the result of dice. If the token passed through the Go Tile, the player receives a salary.

6.2.3 Landing on Tiles

After a player moves their token to a tile, they perform the actions associated with that type of tile.

6.2.3.1 Landing on a Property Tile

When a player lands on a property tile, the property’s Title Deed Card is displayed. The possible actions on this tile are determined by the state of that property.

If the property is owned by another player, the current player pays rent to the owner of the property.

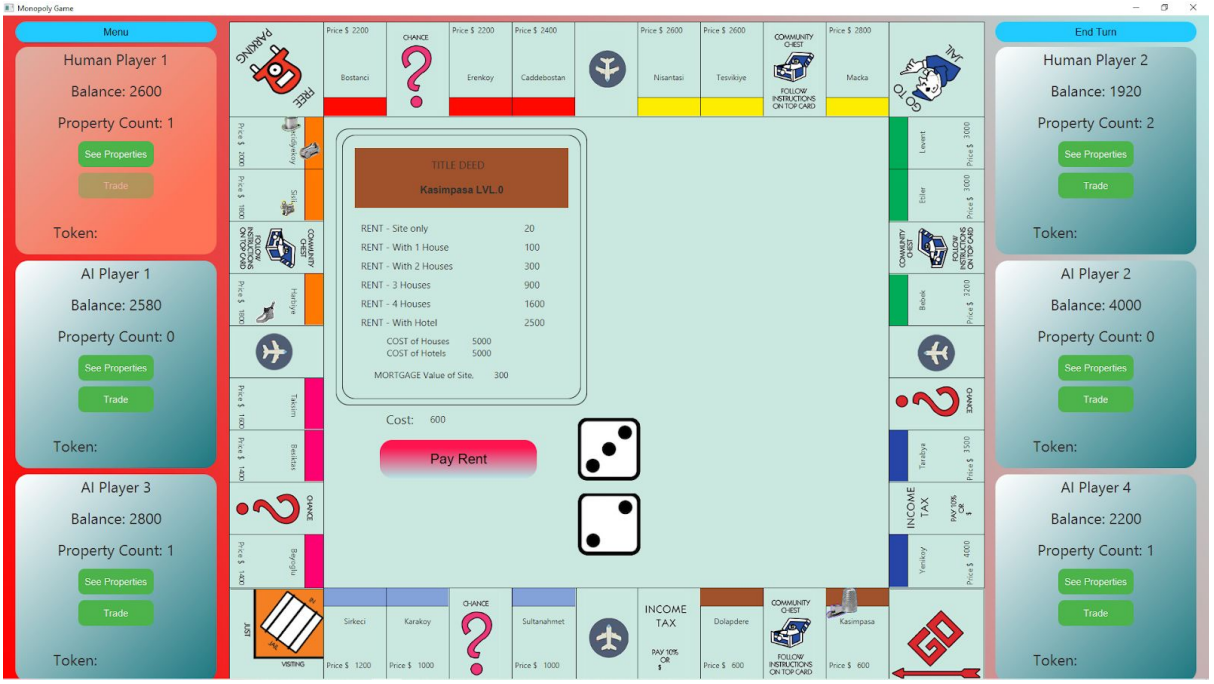

Figure 7. Landing on a property tile, showing the pay rent functionality

If the property is not owned, the Title Deed Card of the property is presented to the player.

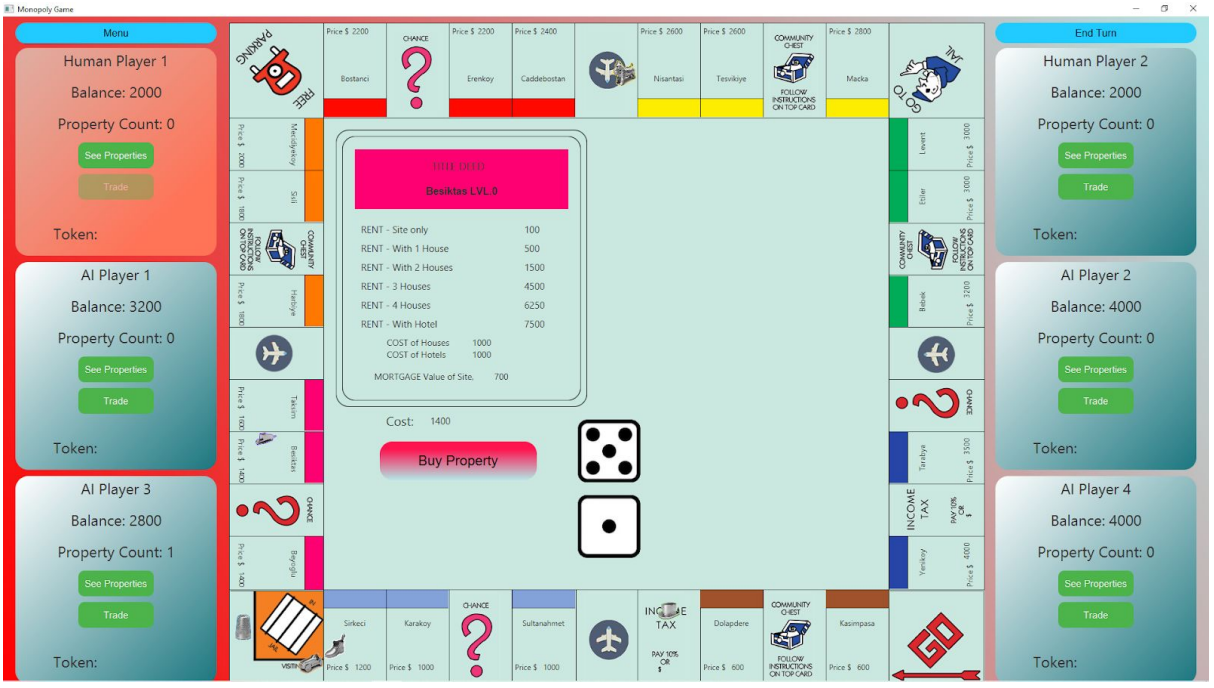

Figure 8. Landing on a property tile, showing the buy property functionality

The player can buy the property by clicking the “Buy Property” button on the screen.

6.2.3.2 Landing on a Card Tile

There are two types of card tiles on the board: Chance Card tiles and Community Chest Card tiles. When a player lands one of those tiles, they draw a card from the corresponding card deck. After the card is drawn, the instruction on the card is presented to the player. The player can perform the action by clicking the “Apply” button.

Figure 9. Landing on a card tile, to give an example the Chance card is displayed.

6.2.3.3 Landing on a Teleport Tile

There are two pairs of teleport tiles on the board. When a player lands on a teleport tile, they move their token to the corresponding teleport tile by clicking the “Teleport”

button.

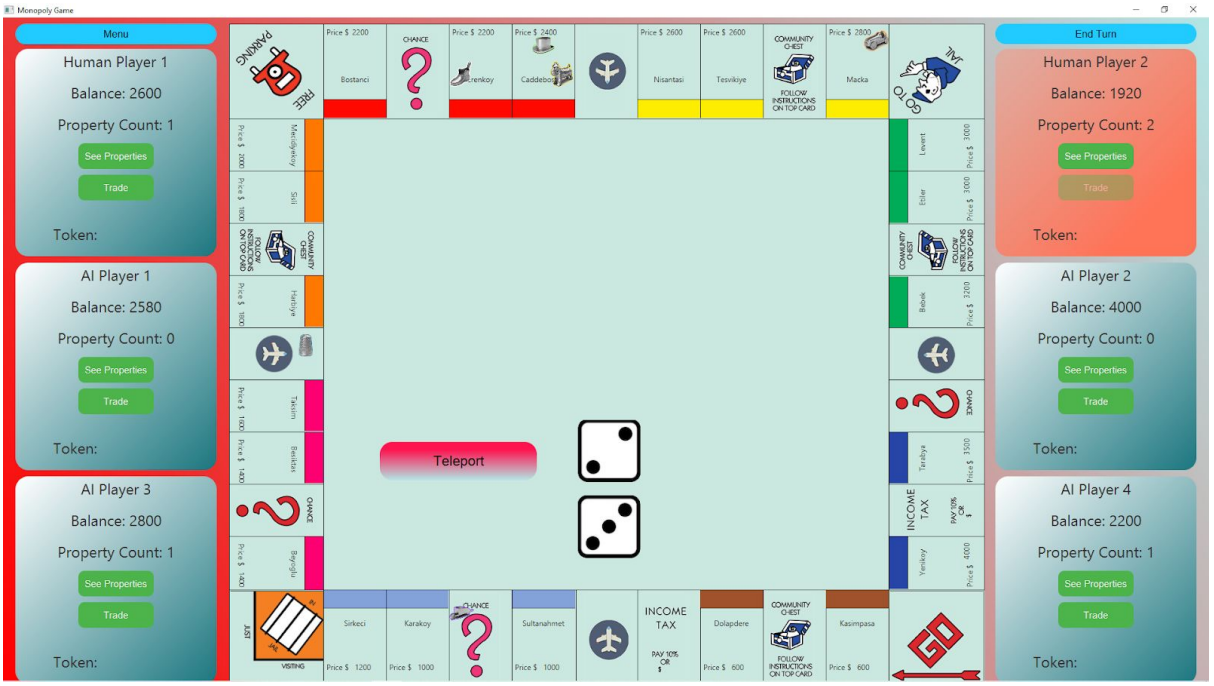

Figure 10. Landing on a Teleport tile, as it can be seen Human Player 2 is at the Teleport tile. The teleport button is displayed.

6.2.3.4 Landing on the Go To Jail Tile

When a player lands on this tile, they move their token to the Jail Tile. The player can not move their token until they bail out of jail.

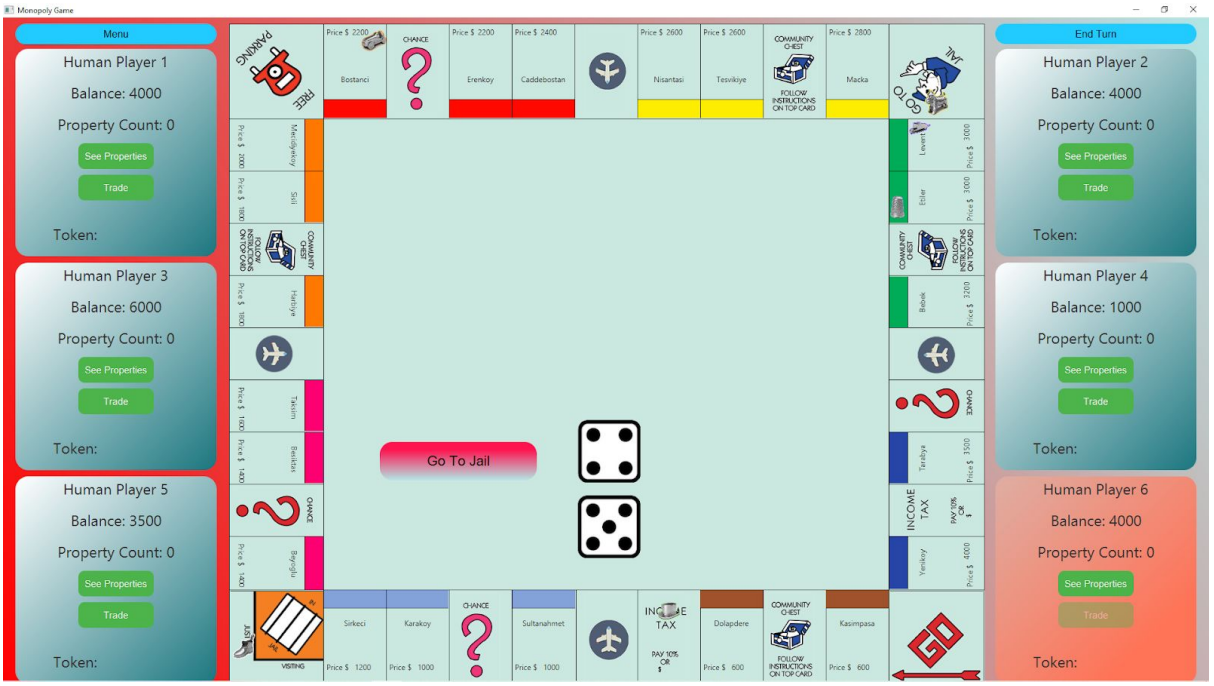

Figure 11. Landing on the Go To Jail Tile, the Go To Jail button is displayed.

6.2.3.5 Landing on the Jail Tile

If a player's token lands on this tile after rolling the dice, no action is taken. However, if the player lands on this tile by landing on the Go To Jail Tile first, they must stay in the jail until they bail out.

6.2.3.6 Landing on an Income Tax Tile

When a player lands on this tile, they must pay an amount to the bank. They can either pay a fixed amount or a fixed ratio, which is an option chosen by the player.

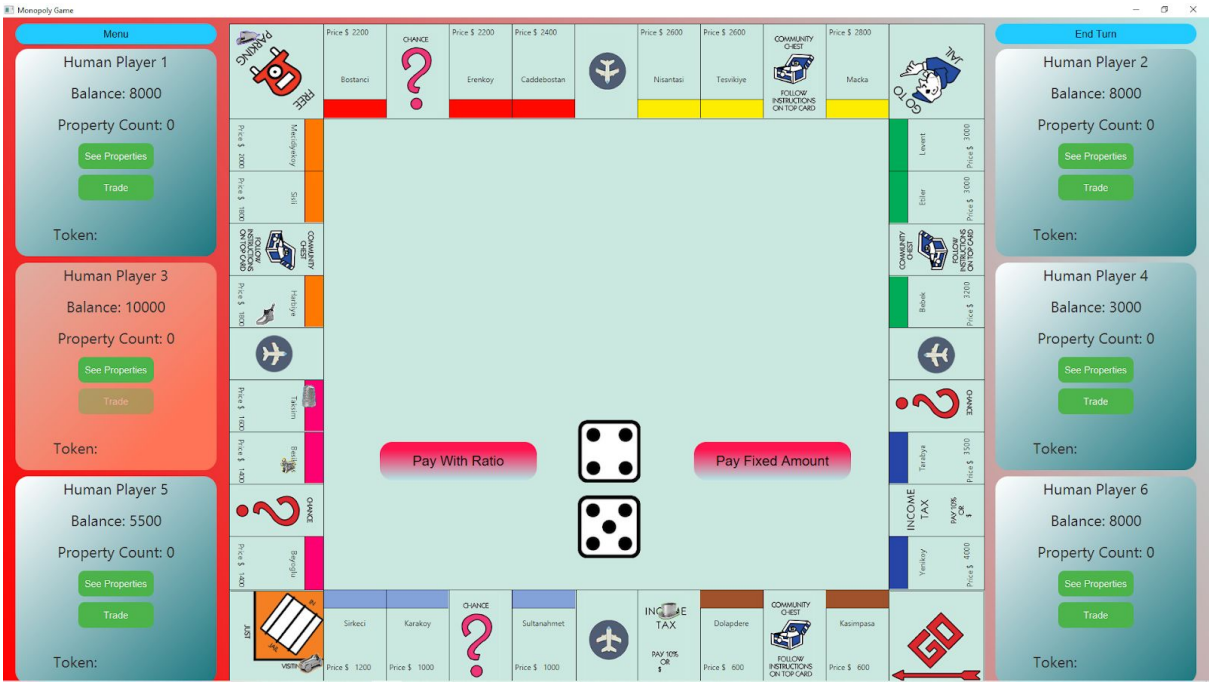

Figure 12. Landing on an Income Tax Tile, here the player must choose between paying with ratio or paying with a fixed amount. This option is applied throughout the game.

6.2.3.7 Landing on the Free Parking Tile

When a player lands on this tile, no action is taken.

6.2.4 Bailing Out of Jail

If the player was sent to jail by landing on the Go to Jail Tile, they must bail out of jail before moving their token. The player have three options to bail out of jail:

- Pay bail bond
- Try to roll double dice
- Use bail out of jail card (If the player has any)

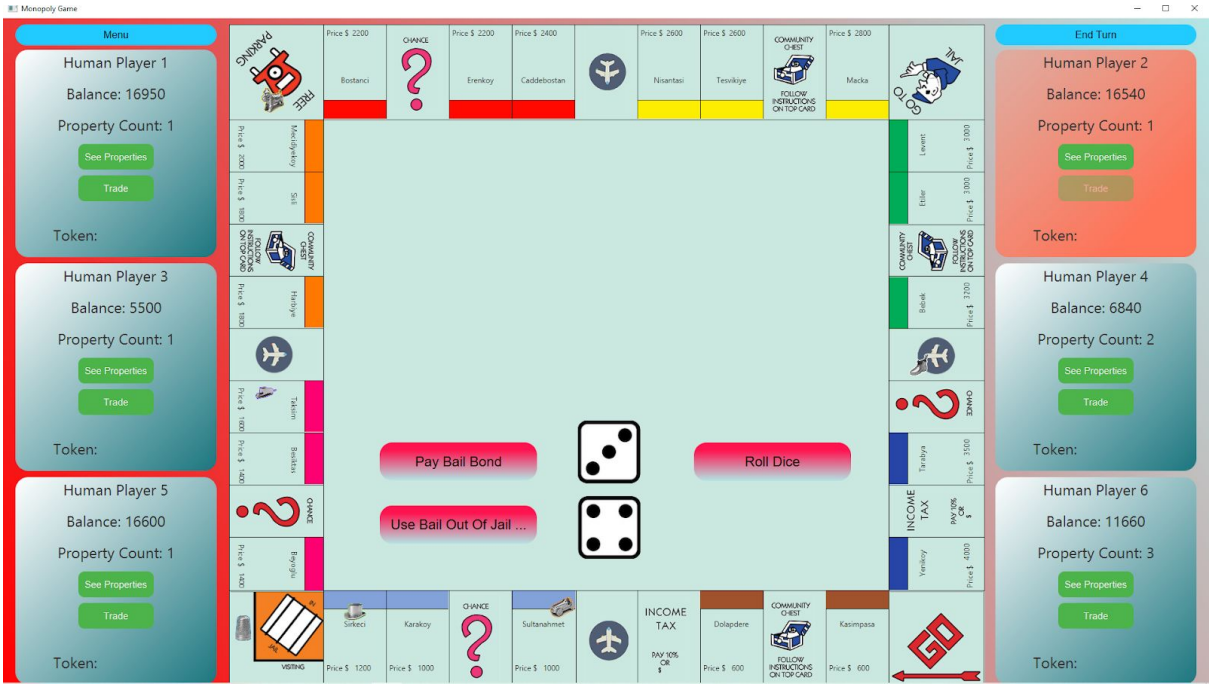

Figure 13. Bailing Out of Jail screen, Pay Bail Bond, Use Bail Out of Jail Card, and the Roll Dice buttons are displayed.

6.2.5 Seeing a Player's Information Card

A player's information card can be opened by clicking the "See Properties" button.

Figure 14. Information Card screen, See Properties button is clicked to access the player's information card.

When the button is clicked the player can upgrade, downgrade, mortgage and remove the mortgage of their properties, and also they can declare bankruptcy.

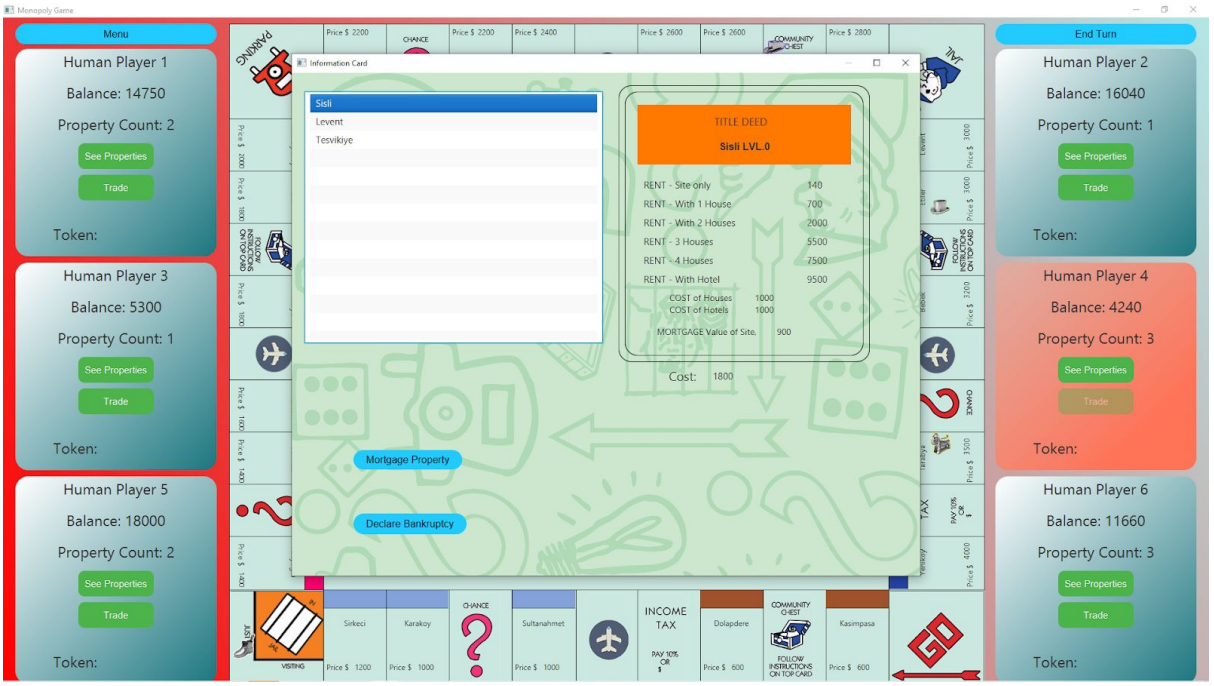

Figure 15. Information Card screen, actions described above can be done in this screen.

6.2.6 Performing a Trade

A player can initiate a trade with another player by clicking the “Trade” button.

Figure 16. Performing a trade screen, as it can be seen the trade button is indicated with the circle surrounding it.

Then, a trade screen appears on the game screen.

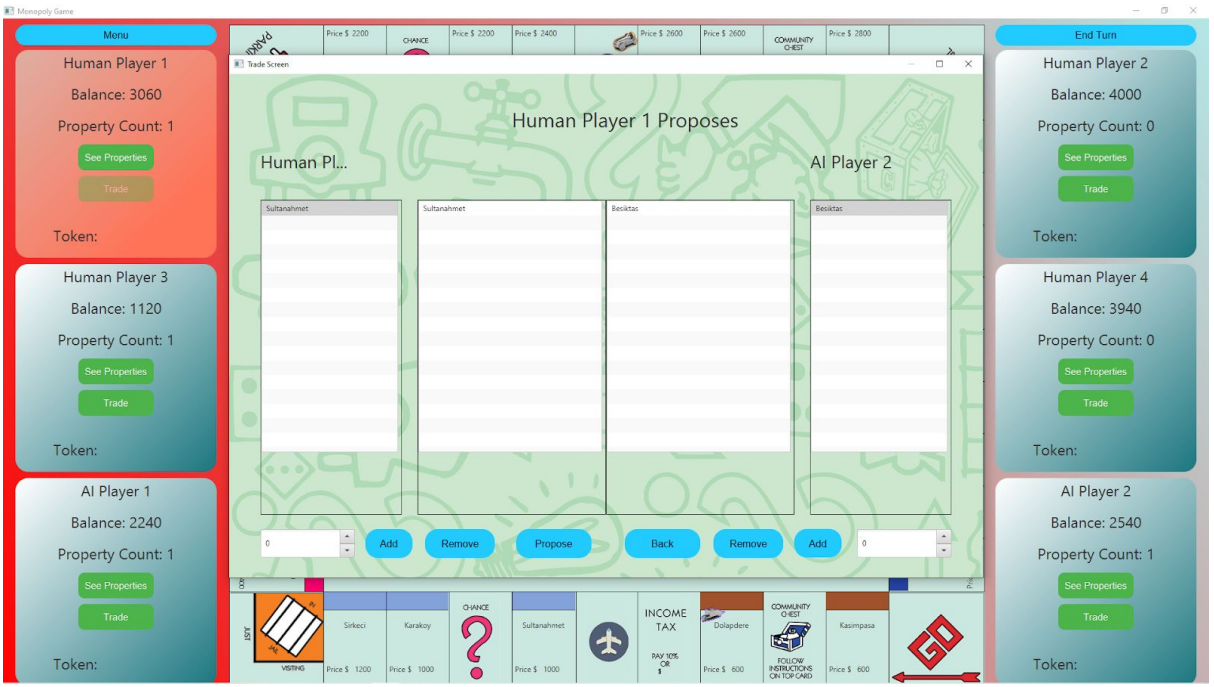

Figure 17. Trade screen, in this screen the players see the trade offers and conclude the trade operation.

On this screen, two players can offer their properties and money to each other. If the other player is an AI player, they accept or decline the offer automatically. AI players do not initiate trade themselves. If the players are both human, the second player also needs to confirm the trade after the first player's proposal.

7. Conclusion

In conclusion, we managed to implement a game that we are happy with. There are a small number of functionalities and features missing as described in section 2.3, however, in the end we managed to implement a game that is visually beautiful, reliable and fun to play with friends or against AI. If we had more time, we could have implemented the missing functionalities as well.

Obviously, there is room for improvement, as it is with any project one does when he/she is learning. However, despite the difficulties, this project has taught us many valuable lessons and we managed to implement a nice game that utilizes design patterns, concepts and tools we were taught in this course.