Bilkent University Department of Computer Engineering

CS 319 Term Project

Final Report

Bilpoly (Monopoly Bilkent)

Section 3 Group 3H

Project Group Members

1. **Ömer Ünlüsoy** 21702136
2. **Arda Akça Büyük** 21802835
3. **İrem Tekin** 21803267
4. **Ece Ünal** 21703149
5. **Furkan Ahi** 21501903

Contents

1. Implementation Process

We started writing the code part of the project when the feedback of the first versions of the analysis and design reports were received. During the development phase of the project, we used various sources:

- First, we decided to use a common IDE and decided to use IntelliJ. This is because it has an advanced and useful interface and the GUI operation can be done relatively easily. In addition, our work has become easier because we use the screen builder it contains while doing these operations. The "GitHub" plugin it contains made it very easy for the group to work synchronously.
- Google Drive, Google Document, MS Word Online and GitHub Desktop have been very useful with the reporting, storage and synchronous work phases of the project.
- While Zoom was used in the early stages of the communication of the group members with each other, this application was seen as inefficient in the following periods and Discord meetings started to be held and Discord was found much more useful.

Different strategies were followed in the report writing and coding parts during the project tasks. In the report part, while assigning our tasks to people piece by piece, in the coding part, everyone focused on the subjects s/he was more familiar with and worked together. Sub-working groups joined the other group in order to work in sync with the other sub-group in which they made a certain progress, and ideas were shared in places. To summarize briefly, the GUI part of the project was created in the light of the previously prepared mockups, while others are working on the back-end part. As the subtasks were over, these parts were put together appropriately.

Various exchanges were made for the game logic and new / changed rules. Although many very nice and original new ideas came up, it was important that the project was completed on time, as we had limited time. That is why the most practical and efficient ideas in implementation came to the fore and were added to the project. As a result, a project was developed by staying true to the trade-offs mentioned in the design report.

The reports we wrote in the development of the project were very helpful and made the coding process much more planned and easier. The coding was largely faithful to the features we reported, as we know that one of the objectives of the course is to write reports and evaluate it appropriately. The design patterns (Decorator and Strategy Design Pattern) we learned in the last part of the course and added to the second iterations enabled the project to be partially reshaped. Thus, a more regular and readable pattern came out.

As a result, we were able to apply most of the features we mentioned in the reporting to the project. While carrying out these, we adhered to the functional and non-functional requirements written in the reports as much as possible. The problem for us was that our test cases were very limited due to time constraints and we could not evaluate all possible bugs. However, if a similar situation is experienced in a real situation, similar bugs can be easily fixed in the process after adequate tests are carried out and user feedback is evaluated.

2. Design Changes

During the development phase of the project, some major or minor changes were made in the process:

- We aimed to provide a more lively and fun game experience by making minor changes from the UI part of the game board.
- We changed our way of using ScreenManager and made the change of managing all screen operations from a single center.
- We aimed for a more competitive game by increasing the prices and rents of the cafes.
- We made arrangements in some subsystems by using the decorator and strategy design patterns we learned. This gave us a more streamlined code and easier implementation.
- We split our classes into subsystems. This provided a smoother working environment for everyone. While making this partition, we were inspired by many of the design patterns we covered in the lesson.

3. Lessons Learned

3.1. Technical Issues

- First of all, it was ensured that everyone learned “GitHub” at an average level in order to be able to communicate with each other in reporting and synchronizing the project. The Lab, which was conducted in the course, made a great contribution to this.
- When we decided to do the project, we decided to use JavaFX with a co-decision but no one in the group knew exactly the JavaFX logic. That's why we spent most of our time learning JavaFX, using the knowledge we learned in IntelliJ IDE and drawing GUIs in programs like Photoshop. This was the hardest and longest part of the project. Since we have limited knowledge, even the smallest mistake may drive the project stuck. We understand that everyone should have at least average knowledge about the frameworks to be used before starting the project.
- In GitHub, merging is not as simple as it seemed for us as the beginners. After a few tricky mistakes, we decided not to use the branching process any longer to ensure the project's safety and no further waste of time (Undoes, stashes, conflicts…).
- There are really no definite limits on the design patterns to be used, and each one of them makes things easier in some way. Although we were quite confused in the beginning, we saw many benefits at the end of the project.

3.2. Team Work

- First of all, we once again understood that communication is a must. Workload sharing, people's level of knowledge and learning to what extent they can apply what they know to the project were tasks that had to be done before the coding stage. GitHub and Discord have been our main assistants to increase our efficiency in this regard.
- Communication, meeting order/time, work sharing and commits improved as the project progressed.

4. What We Did, Did Not

Features we manage to implement with almost no bugs:

- Main Menu
- Changing Background
- Playing Music (setting the volume)
- UI Navigation
- Initialization of Assets (Players, Pawns, Initial Money, Time Mode)
- Atalar's Room
- Dice UI
- Double dice check
- Card Mechanism (Go To Cards, Pay Cards, Earn Cards, Go To and Pay Cards)
- Chance Cards and Rector's Whisper Cards and their workflows
- Buying Buildings
- Paying Rents
- Buying all color set
- Pawn Movement Mechanism
- Scalability and Resize of Window

Features we could not manage to finish:

- Although most back-end is finished and ready, we could not design some UI components such as Buy Bilka and Buy Starbucks, therefore we couldn't connect the back-end with UI.
- We implemented animation for pawn movement but it runs with minor bugs so we comment on it.
- Mortgage process UI
- Bankruptcy UI
- Uploading Music and Background
- Sound FX
- CS Mode

5. Each Members Contribution

Ömer Ünlüsoy:

Reports:

- Sequence Diagrams and Descriptions
- Use Case Descriptions
- Use Case Diagram (Everyone)
- I planned first meetings
- Object and Class Model (Everyone)
- Navigation Path (Everyone)
- Subsystem Decomposition
- Game Logic Subsystem Design
- Game UI Subsystem Design
- Final Object Design
- Packages Description

Implementation:

- I mainly worked on the backend.
- Game Logic Subsystem Design
- Game UI Subsystem Design
- I coded several entity classes
    - Cafe,
    - CardDeck,
    - Player,
    - PlayerDeck
    - Land
- I designed and coded Player Turn Mechanism.
    - playGame()
    - playTurnPreDice()
    - playTurnPostDice()
    - executeLandable()
- I designed and coded Pawn Movement Mechanism.
    - movePawnImage()
    - initializePawns()

- I bound Main Menu with the backend.
    - constructGameManager()
- I handled resize and scalability issues.
    - boardWidth, boardHeight.

İrem Tekin:

General:

- I wrote meeting logs

Reports:

- Use Case Diagram (Everyone)
- Activity Diagram
- State Diagram
- Object and Class Model (Everyone)
- Navigation Path (Everyone)
- Application Control Subsystem Design
- Application UI Subsystem Design
- Final Object Design
- Purpose of System Section
- Design patterns in Design Report
- Game rules in Analysis Report
- Introduction parts
- Game overview

Implementation:

- Application and Game UI parts.
- Chance and Rector's Whisper cards
- Dice images
- Anchoring and scaling
- I did the controllers and FXMLs of the game with Ece.
- Popup management and data flow between the controllers and user interface)
- I prepared
    - How To Play page
    - Player Selection Page

- Credits Page
- Pause menu popup
- Card popups
- Land and cafe popups
- Music

Arda Akça Büyük:

Reports:

- Sequence Diagrams and Descriptions
- Mockups and Descriptions
- Use Case Descriptions
- Use Case Diagram (Everyone)
- Object and Class Model (Everyone)
- Navigation Path (Everyone)
- Game Logic Subsystem Design
- Game UI Subsystem Design
- Final Object Design

Implementation:

- Mainly Backend
- Game Logic Subsystem Classes (Model & Controller Classes & their implementations)
- Design Patterns for Card and FunctionalPlace Classes
- GameManager Class (main game loop, implementation of rules)
- File Management (Reading Buyables and Cards from files)
- Assets of the game (images that been used, txt files...)

Ece Ünal:

Reports:

- Use Case Diagram (Everyone)
- Nonfunctional Requirements
- Activity Diagram

- State Diagram
- Object and Class Model (Everyone)
- Navigation Path (Everyone)
- Application Control Subsystem Design
- Application UI Subsystem Design
- Final Object Design
- Design Goals

Implementation:

- Timer logic and UI
- GameScreen controller (Popup management and data flow between the controllers and user interface)
- Navigation between different pages and popups
- I worked at UI of PreGame Settings, Player Selection and Options pages .
- Background image changes
- I did the controllers and FXMLs of the game with İrem.
- I worked mainly at the UI part of the game.

Furkan Ahi:

Reports:

- Object and Class Model (Everyone)
- Mockups and Descriptions
- Functional Requirements
- Nonfunctional Requirements
- Object Design Trade-offs Section
- Analysis and Design Summary
- Design Goals
- Final Report

Implementation:

- Assets of the images
- Some cards txt files
- Cost of lands, starbuckses, bilkas etc.
- How to play tex

6. User Requirements

6.1. System Requirements

- Java 15
- Java SDK

6.2. Install Guide

→ Go https://github.com/ardaakcabuyuk/CS319_3H_Monopoly

→ Clone the project *Bilpoly*

→ Open the project with IntelliJ IDE and run it. (We do not have an executable for now)

6.3. User manual

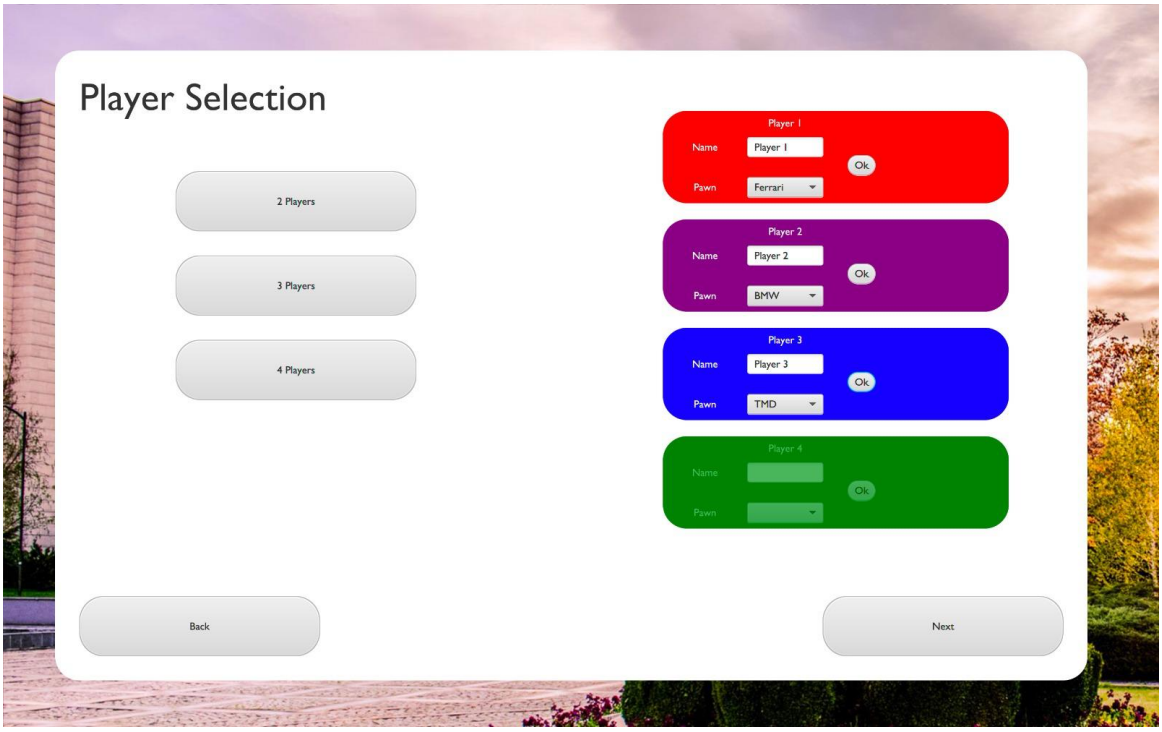
Main Menu

This screen welcomes the user.

- "Play Game" directs the user to the Player Selection and Pre-Game Settings which is required to start the game.
- Before starting the game, candidates can see how the game will be played and the rules by clicking the "How to Play" button.
- There are 3 main options on the "Options" screen. These are:
    - Music level
    - Setting the game background
    - Setting Music level.
- The "Credits" button opens a window with very brief information about game developers.

- "Quit" button terminates the game.

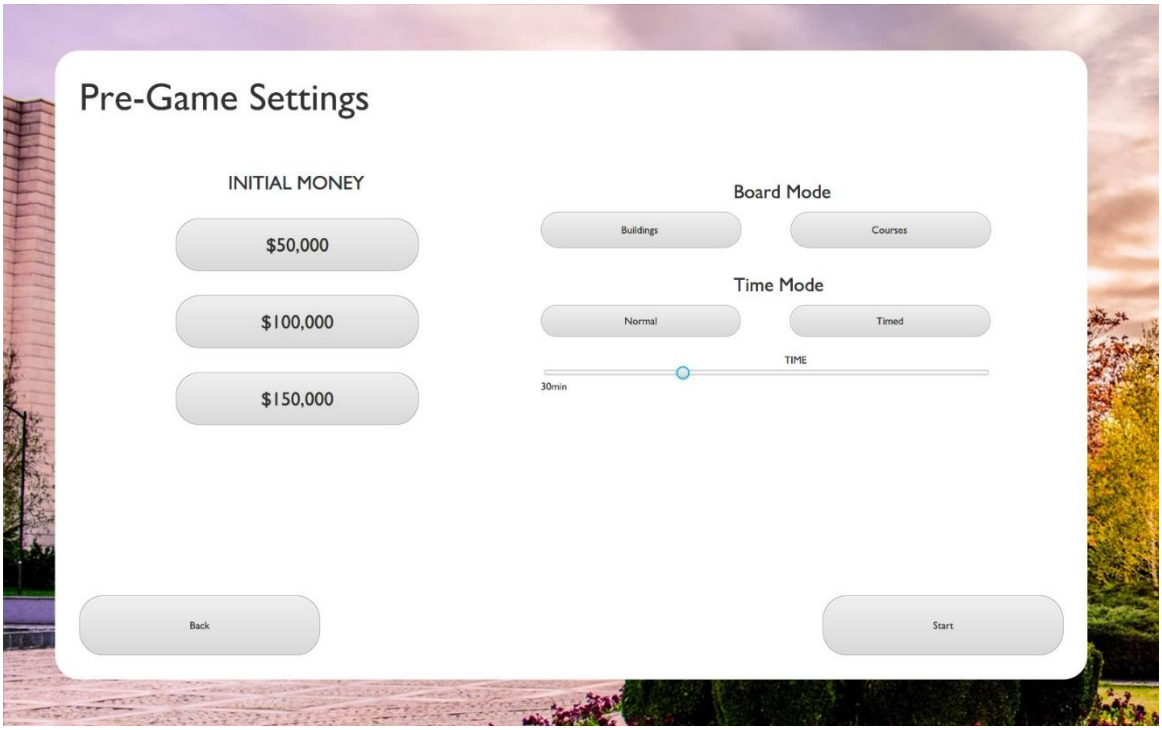
Player Selection Screen



After the user clicks “Play Game” in the main menu, the Player Selection screen appears.

- On this screen, the user is asked to choose the number of people who will play the game, their nicknames and Pawn.
- Once you have completed the entries, you are ready to start the game.
- Click the "Next" button.

Pre-game Settings

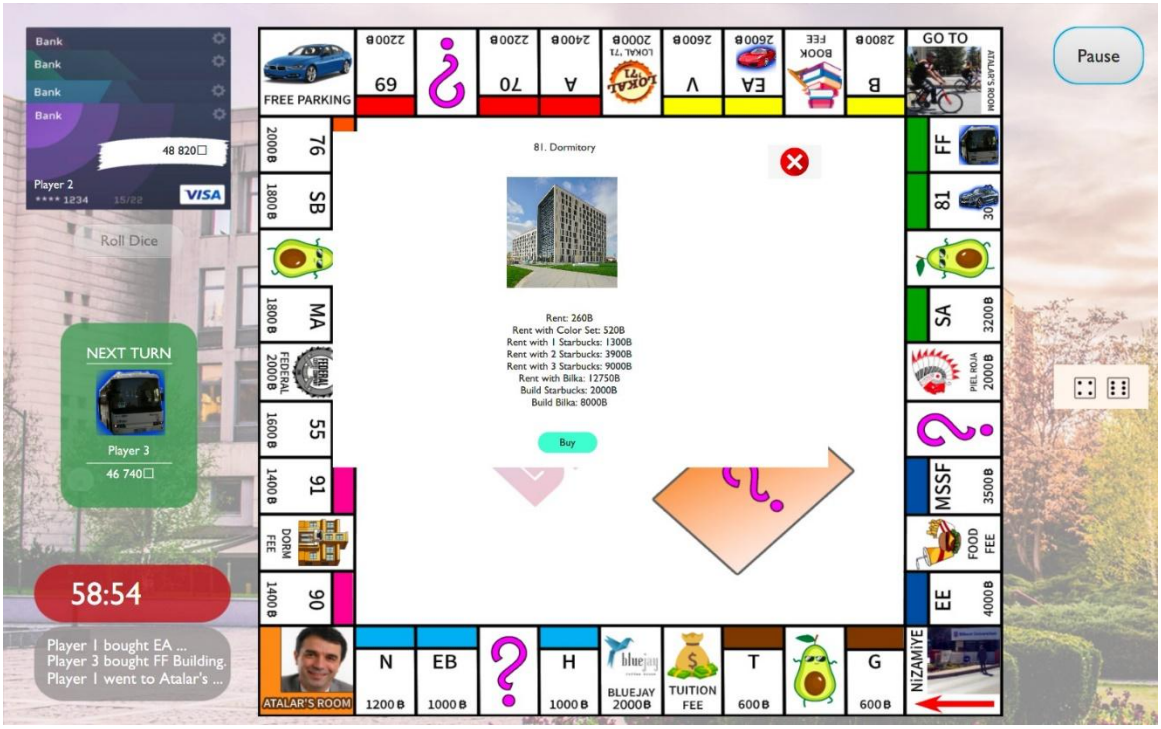


On Pre-Game Settings Screen, players are asked to make 3 main choices:

- Initial money
- Board type
- Game mode

You are now ready to start the game.

Game Screen



This is the game screen of the project. 8 main elements stand out on this screen:

1. Each player has a credit card and credit cards are listed in player order. These cards show the current money of the player.
2. The roll dice button simply automatically handles the roll of the dice.
3. In the "Next Turn" box, the information of the next player is available.
4. The "Timer" box shows the remaining time. Empty if normal mode is being played.
5. The "History" panel is a kind of reminder that shows the users latest moves in the game.
6. The game board is the game center where the property, pawns and all other items of the game are displayed.
7. The "Main menu" button ends the game without saving and takes the players to the main menu.
8. With the "Pause" button you can pause the game and make some adjustments:
   - You can adjust the music level,
   - You can revisit the gameplay and rules.

To keep the user experience at a high level and for a smoother game, we have automated all transactions such as rent payment, property purchase and luck card withdrawal. All you have to do is roll the dice and rely on your luck. You can see the details in the pop-ups :)

7. References

JavaFX. [Online]. Available: https://openjfx.io/. [Accessed: 28-Nov-2020].

“Package java.io,” *java.io (Java Platform SE 7)*, 24-Jun-2020. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html. [Accessed: 28-Nov- 2020].