CPSC 491 - Senior Capstone Project in Computer Science

Department of Computer Science and Engineering

California State University, Fullerton

Jack Loague

Abstract

Video games, as well as board games, are a source of entertainment for many all over the world. This capstone project brings these two mediums together by creating a video game representation of the board game Monopoly, built using the Java programming language. It is hoped that this project can showcase well-written object oriented source code, proper software management using Scrum related principles such as backlog documents and sprints, and clear documentation of the development of the software.

Table of Contents

1. Introduction

One of the most popular branches of software engineering is the development of video games. The industry is one of the most popular sources of digital entertainment in the world. When it comes to physical entertainment, board games are another huge medium. This project aims to combine these two types of games, by creating a digital representation of the existing board game Monopoly, with several added features to incentivize the use of this program over the existing physical version. Reasons for developing this project include an interest in video games, as well as creating an application with a graphical user interface. It is also hoped this project can strengthen skills in object oriented programming.

2. Project Plan

The development of this capstone project will first start by identifying the overall features needed in the software. This project aims to use tenets of the Scrum methodology of software development to aid in its development. Within the Scrum framework, various artifacts are produced throughout its process to help define requirements and organize work into a schedule. To start, a product definition document will be created which defines the main features of the game. These features will then be broken into user stories within a product backlog document.

The architecture and UX of the game will then be designed through a series of diagrams and images, to give a clear definition of how the software should function and what its gameplay should look like at all stages. Once requirements and design have been determined, programming will become the main focus of the project.

During the programming phase, to decide what will be worked on and when, the development of this project will be split into two-week intervals, known as a sprint. During a sprint, a healthy amount of user stories from the product backlog will be chosen and broken into more specific, manageable tasks, to be worked on for a two week period. The various tasks that are to be worked on for a specific sprint are organized in a sprint backlog, to identify which tasks need to be started, which are in progress, and which have been completed. Any tasks that are not finished within a sprint will be pushed to the next sprint.

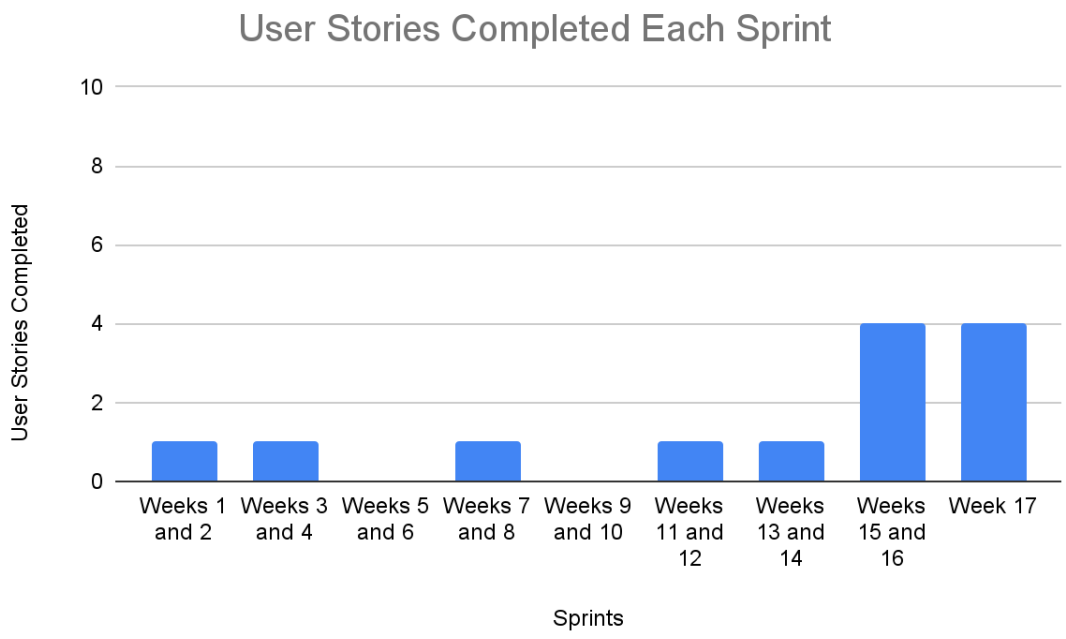
Once a decent amount of the software has been created, testing may begin to ensure it meets the requirements laid out at the beginning of development. Once the software has almost been completed, and the various sections of this capstone report have been finished, the user manual for the software and conclusion for this report can then be written for this capstone project.

3. Metrics

The progress of this capstone project is tracked using certain metrics listed below.

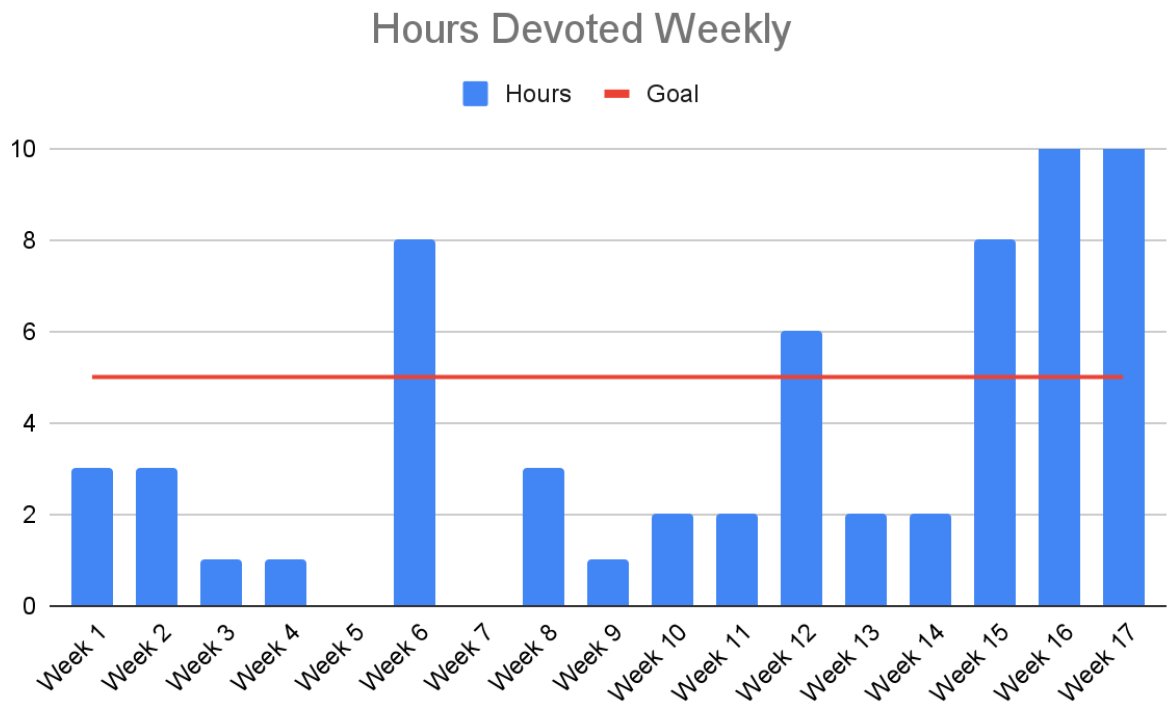
3.1 Number of User Stories Completed per Sprint

This capstone project makes use of the various tenets of the Scrum methodology of software development, including the creation of a product summary and backlog (with a list of features and user stories for the project). Every two weeks of the project's development, a sprint begins, and a healthy amount of various user stories are assigned to be worked on for the duration of that sprint. These user stories are broken into smaller, more specific tasks that are listed in the sprint backlog for the given sprint. As a way to track the progress of this project, the number of user stories fulfilled (meaning all tasks from the user story are completed) during each sprint iteration (or every two weeks), can be represented by the bar graph below.



3.2 Number of Weekly Hours Devoted to the Project

It is hoped that a goal of around five hours of time per week can be achieved during the development of this capstone project. Time dedicated to either the actual coding or the writing of the project's documentation can be considered as time spent towards the project. The number of hours will be tracked during development time, and is another way of measuring the progress of this project, and to ensure that enough time is devoted to its development. The number of hours that have been devoted weekly to this project can be viewed by the visual below.



4. Maintenance Plan

4.1 Patches

Bugs identified in the software will be fixed as quickly as possible. Once they are fixed, the software will be updated for users to download. These fixes will not be considered new versions of the software, but rather simply just be considered patches.

4.2 New Versions

If a new feature is added to the software (for example, a profile statistics screen for each game profile may be a feature that will be added after the first version of the software is released), this update can constitute a new version of the game.

5. Requirements Engineering

5.1 Requirements Specification

The requirements of this capstone project are outlined below, split into functional and non-functional requirements.

5.1.1 Functional Requirements

Game Menu

Main Menu

- When the game is started, the main game menu should appear. There will be several buttons listed in this menu for the user to choose from: Profiles, Play, and Options:
    - **Play**: Navigates to a set up screen where users can set up and begin a game. There will be an interface to add/remove users, and select the player profile and token for each user.
    - **Profiles**: Navigates to a profile page where a user can create/delete an existing player profile, and view a list of the personal statistics of each player profile. These personal statistics will include: Total Wins, Total Losses, Most Money in Game, Most Houses in Game, and Most Hotels in Game. Profiles should be saved even after the program is closed.
    - **Options**: Navigates to an options menu where users can control the volume of sound effects, as well as adjust the control scheme of various functions in game.

Main Game

Start of the Game

- Upon starting a game, the main game board should be displayed, along with the tokens of each player, with all tokens starting on the Go space. The player turn order will follow the order created in the Play menu previously mentioned.

Player Turn

- During a player's turn, they will have three options available to them: rolling the dice, opening a player manager menu, or ending their turn. Each of these may be accessed by either clicking a button or pressing a key:
    - **Rolling the Dice**: A short animation of dice rolling will play, ending with the dice displaying the amount of spaces the player will move.
    - **Player Manager**: A new interface will appear with a list of options that the player may choose from, including: trading with other players, mortgaging or unmortgaging properties, and creating or selling houses and hotels. Each of these options will open up another interface, and can be chosen by the player through clicking a button or moving a pointer onto an option and pressing a key:
        - **Trading**: The interface for trading will include a list of players that are in the current game to choose from. Upon choosing a player, all of the properties, cards and total money of both the current player and the player chosen will be displayed. The current player may then select which properties or cards they wish to trade and which properties or cards they wish to receive. They may also specify an amount of money they wish to

trade or receive. The player who is being traded with will then be given the option to accept or decline the trade, or provide a counter offer.

- **Mortgaging or Unmortgaging Properties**: The interface for mortgaging or unmortgaging properties will display a list of properties that the player owns. Upon selecting a property, the amount of money needed to either mortgage or unmortgage the property will be displayed. The player may then choose to mortgage or unmortgage (if they have enough money) the property by clicking a respective button. Properties with houses or hotels may not be mortgaged.
- **Creating or Selling Houses and Hotels**: The interface for creating houses and hotels will display a list of properties where the player may build or sell a house or hotel. Upon selecting a property, the amount of money required to build said house or hotel will be displayed, along with the amount of money that will be provided by selling a house or hotel. By clicking a button, the player may either create a house or hotel if they have enough money, or sell a house or hotel if the property currently has houses or a hotel. A house may only be created if all other properties in the respective set also have the same number of houses created.

- **End Turn**: Once a player has rolled the dice and followed the directions for the space they have landed on, they may end their turn by clicking this button.

Space Event

- Once a player has rolled the dice and the dice animation has finished playing, their token icon will animate and move the correct number of spaces across the board.
- Once a player has landed on a space, the name of the space will be displayed on screen. Depending on the space type, a different process will occur:
    - **Go Space**: If the player passes or lands on the Go space, they will be rewarded a salary of $200 which will be added to their total money. A message will appear on screen notifying them of this.
    - **Jail/Just Visiting Space**: If the player lands on the Jail/Just Visiting space, nothing will occur.
    - **Free Parking Space**: If the player lands on the Free Parking space, nothing will occur.
    - **Go to Jail Space**: If the player lands on the Go to Jail space, their token will move directly to the Jail/Just Visiting Space. They will now be considered in jail, and may not leave this space until one of three events happen: the player rolls doubles, the player pays a bail of $50, or the player uses a get out of jail free card. An interface will appear for the player each turn they are in jail to choose one of these three options. After three turns have passed, if the player is still in jail, they will be forced to leave jail, either by paying the bail or using a card.
    - **Property Space**: If the player has landed on a property space, a variety of events may occur:
        - **Unowned Property**: If the property is unowned, the price of the property will be displayed, and the player may choose to buy the property if their

total money is greater than or equal to the price of the property. They may also choose to not buy the property. If the property is purchased, it will receive a colored marking to indicate that it belongs to that respective player. Different colors will be associated with each player depending on their player number in the set up screen.

        - **Owned Property**: If the property is owned, the player must pay the specified rent to the owner player. If the current player does not have the funds to do so with their money in hand, they will have to navigate to the player manager menu to raise funds.
    - **Card Space**: If the player lands on a card space, they will receive a card and must follow its instructions. Examples include having to pay a fee, or moving to a specified space on the board.
    - **Tax Space**: If the player lands on a tax space, they will be required to pay a specified amount of money, depending on the space.

Bankruptcy

- When a player does have the funds to pay a debt, they must declare bankruptcy by clicking a button.
- Upon declaring bankruptcy, the player will be removed from the game, and all of their assets, including their money, properties, and cards, will be transferred to the player who causes the bankruptcy. If the player became bankrupt because of a tax space or a card, all of their assets will be transferred to the bank, and all of their properties will then become unowned again for any player to purchase.

End of the Game

- Once there is only one player left in the game, a text screen will appear informing them that they have won.
- The game may then be returned to the main menu through a button. The personal statistics of each player who participated in the game will then be updated.

5.1.2 Non-Functional Requirements

Usability

- The components of the game's interface should be spaced out and appropriately scaled. For example, during the main game phase, the game board should appear in the center of the screen and be the most prominent component.
- It should be very intuitive and easy to navigate through the various interfaces of the game. Button names and menu titles and options should be properly labeled and have clear meanings so the user is not confused.

Performance

- The game will run at 60 frames per second, and should not lag at all during gameplay.
- The game should be responsive and contain essentially no loading time between game states. For example, when clicking a button to start the main game, there should be practically no wait time.

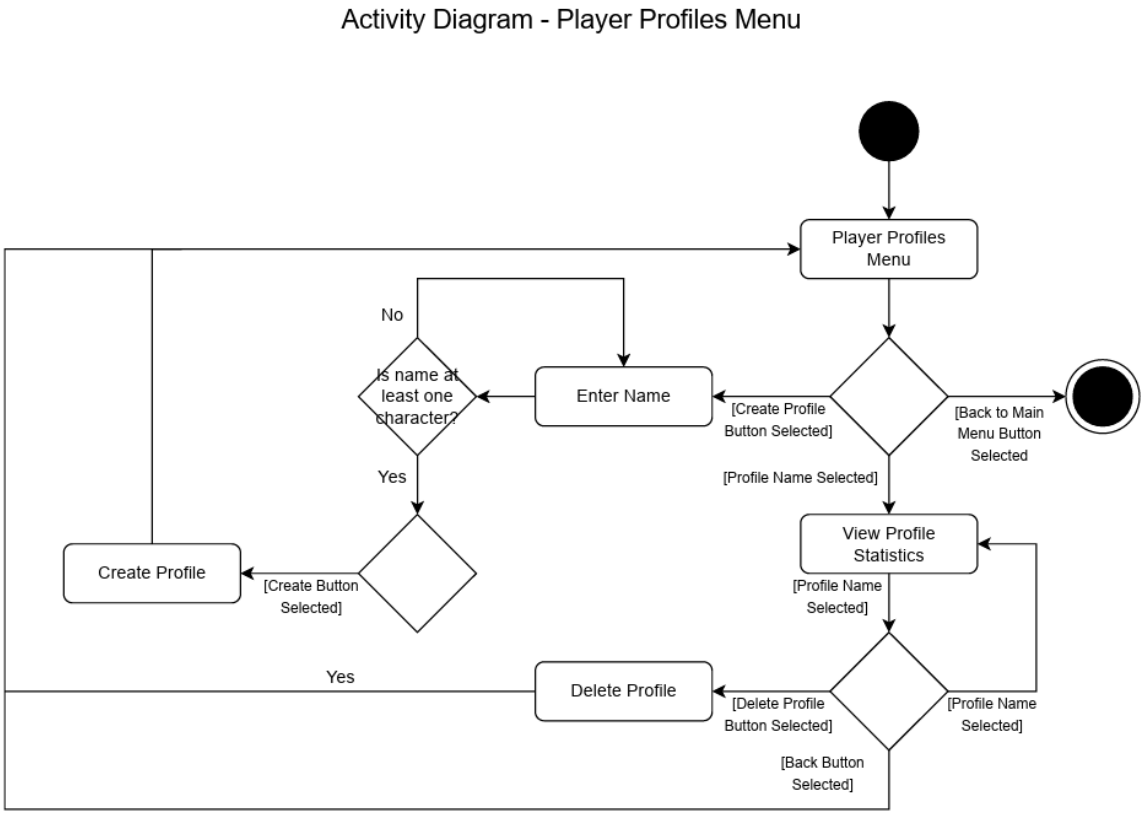
Maintainability

- Each function of the game should have a unit test associated with it. This is to ensure that all game functions still perform correctly if any part of the source code of the software is ever changed.

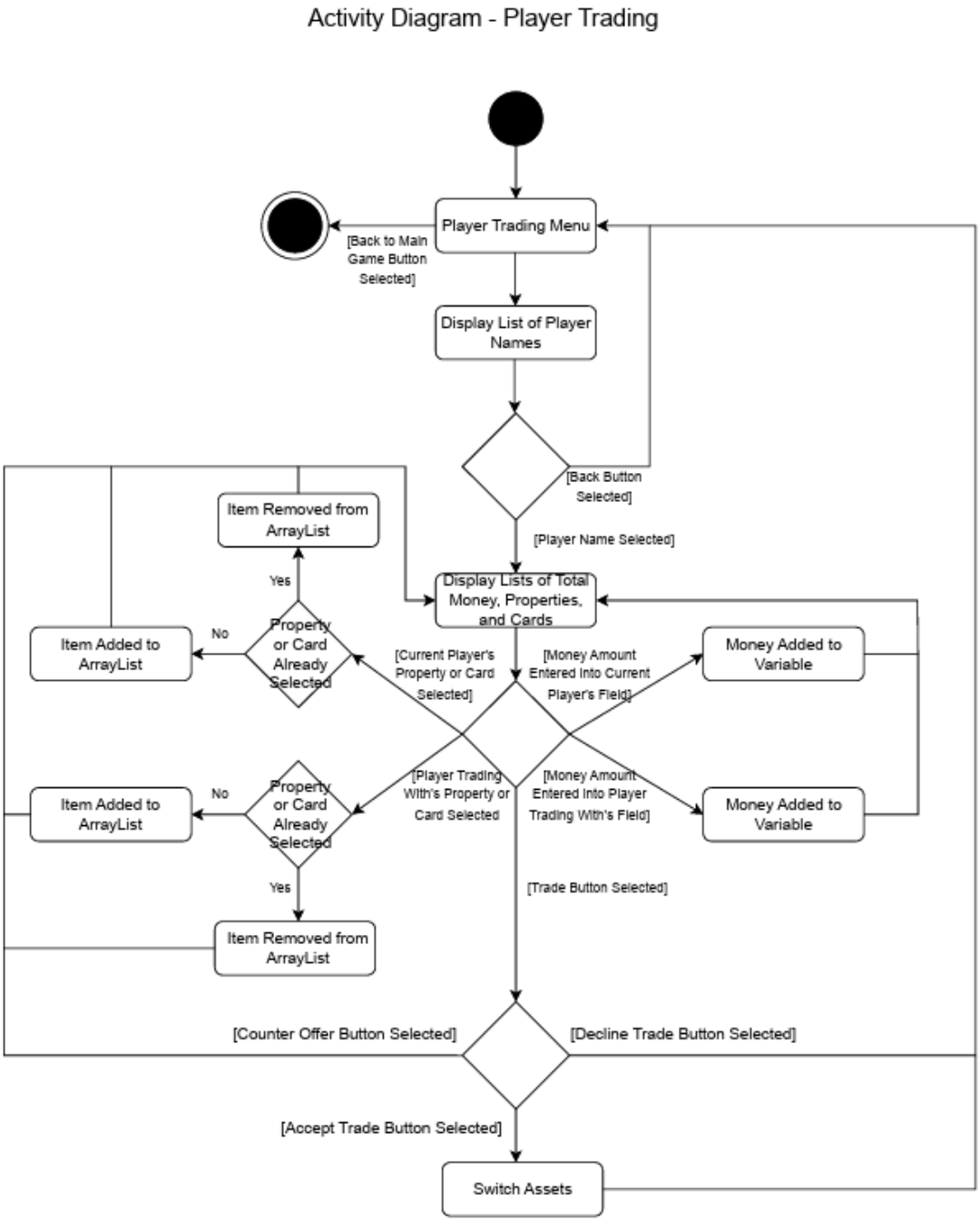
5.2 Modeling Tools and Techniques

To increase understanding of the requirements laid out in the specification section above, a couple of activity diagrams have been created to visualize the flow and logic for some of the interfaces that will be used in game.

5.2.1 Player Profile Menu Activity Diagram



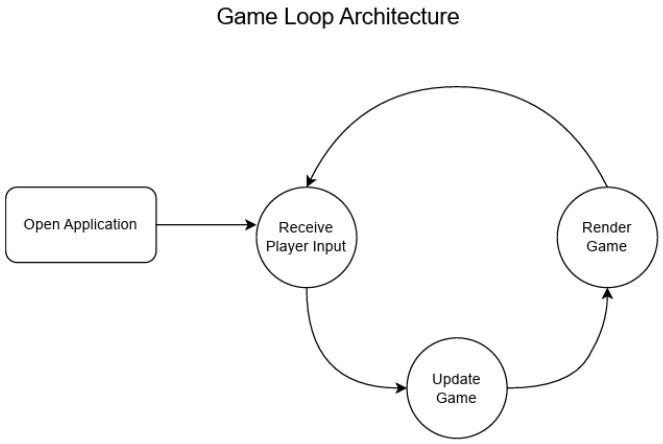
5.2.2 Player Trading Interface Activity Diagram



6. Architecture and Design

6.1 Main Game Loop Architecture

Due to the nature of video games, the software will have to update itself constantly for events such as user input, game state changes, animations, etc. The game will plan to be run at 60 FPS, meaning that the software must update 60 times per second. Because of this, a game loop architecture will be implemented for this application. A diagram can be seen of its design below.



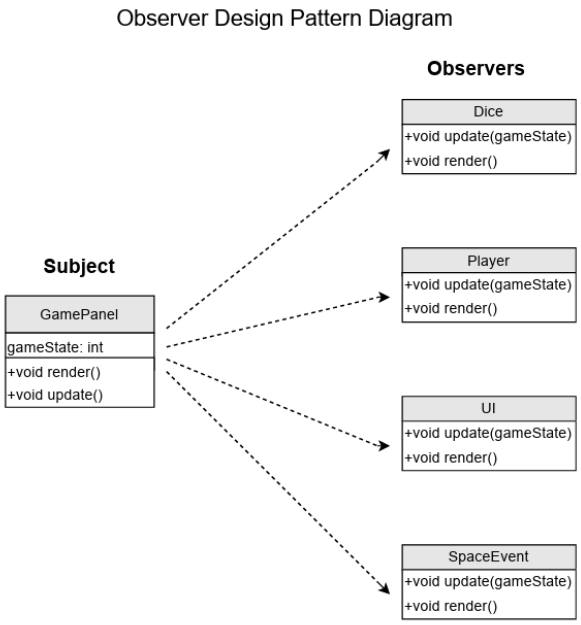
Receive Player Input: This is where user input will be processed. This will be handled through Java classes such as KeyAdapter and MouseHandler for key and mouse input respectively.

Update Game: This is where game logic will be updated, such as the game state (for example, changing from the main menu phase to the main game phase), variables (such as player assets and token coordinate locations), among other information.

Render Game: This is where the graphics of the game will actually be displayed, reflecting changes in the game logic in the Update Game stage of the loop.

6.2 Game Update Architecture

During gameplay, multiple objects may need to update and render depending on the game state. Examples of objects in this software include: players, UI elements (or interactable elements), and space events (which will handle any events which occur when a space is landed on). Because of this, an observer design pattern will be used. In this design pattern, an object, considered the subject, keeps track of a list of dependent objects, and notifies them about any changes that occur to the subject object (Chia, 2021). With this architecture, all game objects can be updated from a central object depending on the current game state.



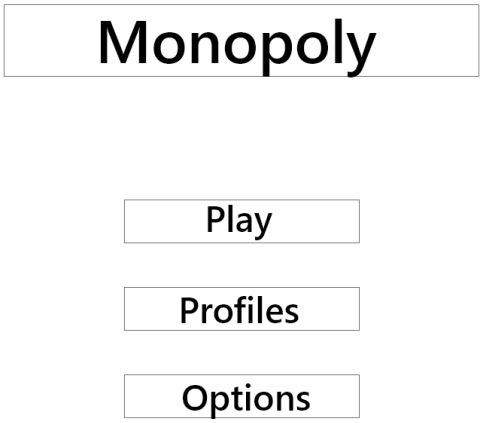
JPanel is a container that can group graphical components together. Because of this, the update and render methods for the entire game will be stored in a JPanel class named GamePanel. This will be the subject class. From here, the appropriate update and render methods of other observer objects in the game can be called depending on the game state.

For example, if the game is in a state where a player should animate across the board after rolling the dice, the update method in GamePanel will call the Player's update method, passing the current game state. The Player update method will then know to move the player's coordinate variables. The render method in GamePanel will then call the render method in Player and update a player's visual location on screen.

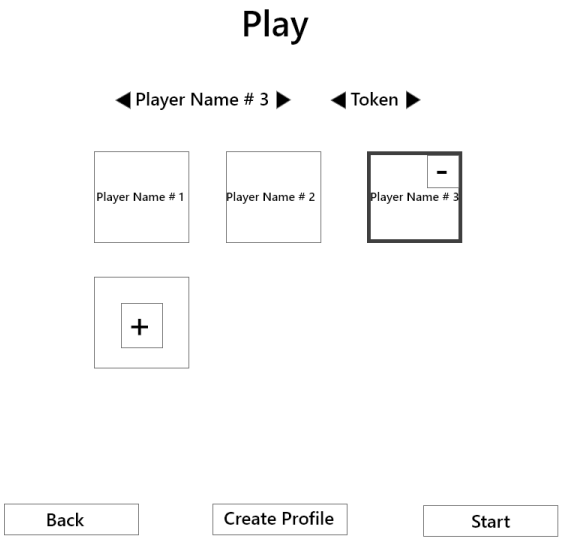
7. UX Design

The UX design of various screens that will be seen throughout the game are documented here.

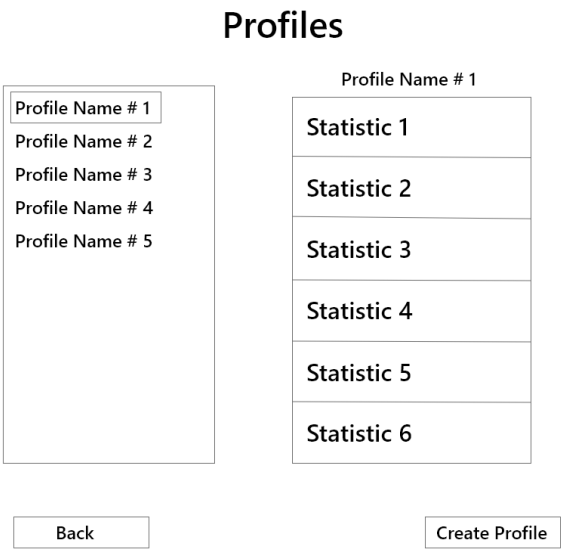
7.1 Main Menu



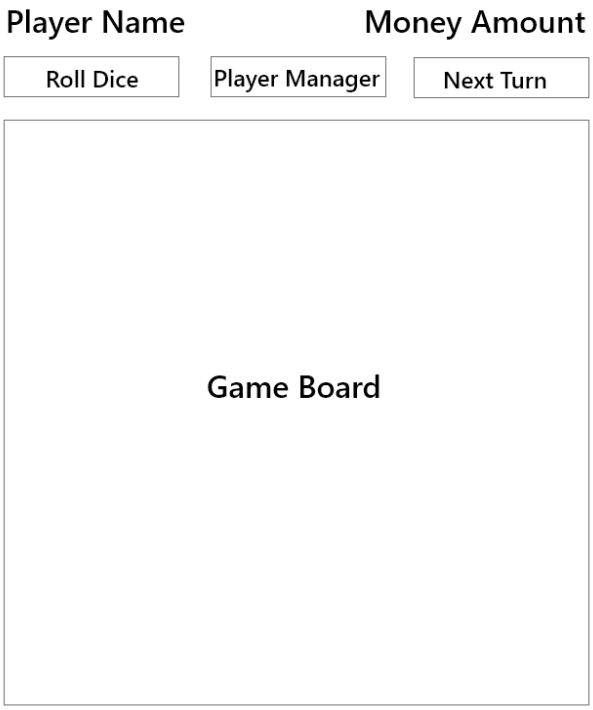
7.2 Play Screen



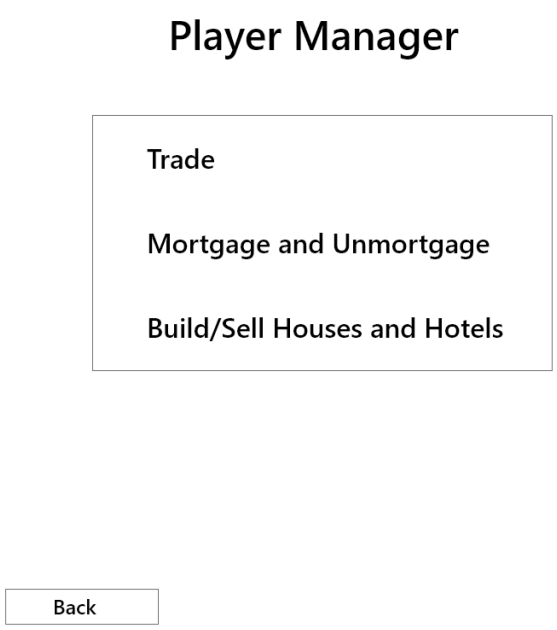
7.3 Player Profiles Screen



7.4 Main Game Screen

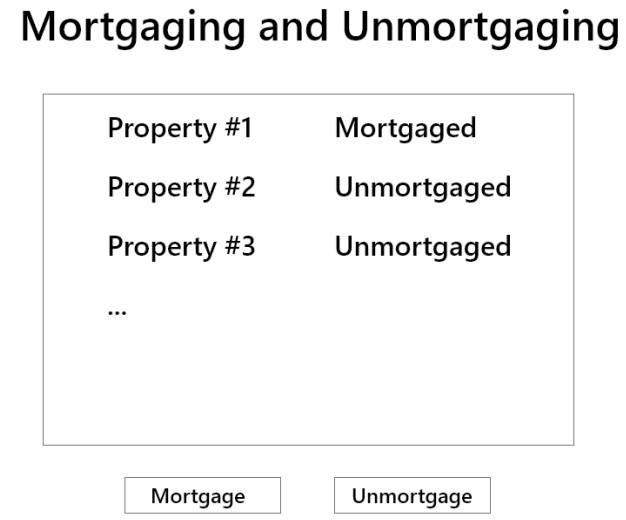


7.5 Player Manager Screen



7.6 Trading Screen

7.7 Mortgaging and Unmortgaging Screen



7.8 Building and Selling Screen

8. Prototyping

A prototype of the software should be available by Week 9 of the project's development. The full scope of the game does not need to be finished. Rather, a player setup screen, the main game board, and the ability for tokens to move around the board by rolling the dice is all that will be required. The UI does not need to be finalized yet either. This prototype should simply show off a simplified version of what the final game should look like.

9. Implementation Methods

9.1 Coding Standards

9.1.1 Naming Conventions

Variables

- Variables should have meaningful names that describe their purpose.
- Normal variables should be written using camel casing (Example: exampleVariable)
- Constant variables should be written in all caps with spaces between words (Example: EXAMPLE_CONSANT)

Functions

- Functions should have meaningful names that describe their purpose.
- Function names should be written using camel casing.

9.1.2 Spacing, Indentation, and Formatting

- All imports should be grouped into code blocks depending on their type (external libraries and local source code files). External library imports should be listed first, followed by local source code file imports, with one line of whitespace between each group. The imports in each grouping should be listed from top to bottom alphabetically.
- There should be one line of whitespace between any previous lines of code and standalone function headers, class headers, class method headers, if statements, and loops.
- The opening and closing parenthesis of standalone functions, classes, class methods, if statements, and loops should be on their own separate line.
- All code within standalone functions, classes, class methods, if statements, and loops should be indented with a single tab.
- Each line of code and each comment line should not exceed 80 characters, and should wrap to the next line if so.

9.1.3 Comments

- A healthy amount of comments should be written throughout the code to explain what is happening.
- All standalone functions, classes, and class methods, should have at least a one line comment written above them explaining their purpose.
- Individual lines of code or related blocks of code that are somewhat difficult to understand should have comments explaining in detail what they do.

9.2 Best Practices

- The use of global variables should be avoided when possible, as they can be altered by any section of the source code, among other reasons.
- Lengthy functions should be avoided when possible, to make the source code more readable.
- There should never be too many layers of nesting, as this can be difficult to follow.
- Long stretches of code should be spaced out to improve readability. This can be done by separating more related chunks of code into blocks.

9.3 Tools for Evaluating

The source code for this project will be developed in the Visual Studio Code text editor. The CheckStyle for Java extension will be used to properly format and style the source code. The extension's formatter will be configured to the Google Java Style Guide, as it is a formatter that is provided with the extension. Formatting will be checked regularly using this extension. The only setting changed from the Google style guide will be extending indentation from two spaces to four spaces.

10. GitHub Repository

The GitHub repository for this project can be found through the link below.

https://github.com/jackloague1/Java-Monopoly-Project

11. Open Source and Third Party Components Used

11.1 Abstract Window Toolkit

Abstract Window Toolkit (awt) is a package for creating graphics and user interfaces (Oracle). This package will be used mainly for visual functions, such as displaying colors and images.

11.2 Swing

Swing is a package also used for creating user interfaces. The main difference between this package and the abstract window toolkit is that swing is designed to work the same on all platforms as closely as possible (Oracle). This package will be used mainly for creating interactable components, such as buttons the user can click on.

11.3 JUnit

Some of the features of this project were tested through unit tests. These unit tests typically test different methods within the game. To accomplish unit testing for this project, the open-source framework JUnit was used. This framework provides a nice way to test different functions of the program and run many different tests on different methods at once. This can be useful in ensuring that various functions in the program still work when the source code is changed.

12. Video Demonstration

Below is a Google Drive link of a video demonstration of the project:
https://drive.google.com/file/d/1lFE6h75SsL2CjxmAH022ZonUynvpINj7/view?usp=sharing

13. Test Plans and Results

13.1 Test Cases

Save and Load Profile Test - Check to make sure that a profile is created in the program is saved and loaded whenever the program is closed and reopened.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
1	1. Delete the profiles.txt file within the saved-data folder of the project, so the program can be started with no saved profiles. 2. Start the program. 3. Navigate to the Profiles menu and create a new profile. 4. Exit and reopen the program. 5. View the profiles list in the Profiles menu.	None.	The profile that was created before the program was exited should be displayed in the Profiles menu.	As expected.

Check if Set-Up is Ready Test - Check that a game can not be started in the Set-Up Game menu until all added players have selected a profile name and a token.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
2	1. Start the program. 2. Create at least six profiles in the Profiles menu. 2. Navigate to the Set-Up Game screen. 3. Set up a game with varying amounts of players. 4. Deselect a profile name or token for at least one player.	None.	If at least one player does not have a selected profile or a token in the Set-Up screen, the Start game button should be grayed out and should not function.	As expected.

Passing Go Space Test - Check that a user can collect $200 salary when they pass the Go space, and that their token can continue moving after clicking the “Ok” button on the Go space message.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
3	1. Hardcode a player’s position to a space where they can pass the Go space in one roll. 2. Hardcode dice results in a way where a player will pass the Go space. 3. Start the program and begin a game. 4. Roll the dice and wait for the player’s token to land on the Go space. 5. Press the “Ok” button at the displayed Go space message.	None.	The current player’s token should continue moving to their destination space after clicking the “Ok” button for the Go space message.	As expected.

Landing Directly on Go Space Test - Check that a user can collect $200 salary and is able to end their turn after landing directly on the Go space.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
4	1. Hardcode a player’s position to a space where they can reach the Go space in one roll. 2. Hardcode dice results in a way where a player will land directly on the Go space. 3. Start the program and begin a game. 4. Roll the dice and wait for the player’s token to land on the Go space. 5. Press the “Ok” button	None.	The current player should be able to press the “End Turn” button after clicking the “Ok” button for the Go space message.	The Go space message keeps displaying after pressing the “Ok” button for the Go space message. The program is stuck in an infinite loop and the player is not able to end their turn (see section 15.1 for the fix for this bug).

	at the displayed Go space message.			

Utility Rent Test - Check to make sure that a utilities rent is either four times the number displayed on dice if one utility is owned, or ten times the number displayed on dice if both utilities are owned.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
5	1. Hardcode dice results to always equal four so each player can land on a Utility space in few turns. 2. Start the program and begin a game with two players. 3. Have player 1 buy each utility property as they come across it. 4. Have player 2 land on each utility property after it is bought by player 1 to see the rent for each.	None.	Player 2 should have to pay four times the number they rolled on the dice when they land on the first utility, and ten times the number they rolled on the dice when they land on the second utility	The rent for each utility property ended up being $0 (see section 15.3 for the fix for this bug).

Jail Test - Check to make sure that players can head to jail.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
6	1. Hardcode dice results in a way where players can reach the Go To Jail space within a few turns. 2. Start the program and begin a game with any amount of players. 3. Have a player land on the Go to Jail space.	None.	A message should display informing the current player that they must head to jail. The player's token must then appear in the jail space.	As expected.

Jail Functionality Test - Check to make sure that a player who is in jail cannot leave jail until they roll doubles, or pay a bail.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
7	1. Hardcode dice results in a way where players can reach the Go To Jail space within a few turns. 2. Start the program and begin a game with any amount of players. 3. Have each player land on the Go to Jail space. 4. Have the first player who is in jail roll the dice. 5. Have the second player who is in jail pay bail.	None.	The first player should only be able to leave the jail space if they rolled doubles. The second player should be able to leave the jail space after paying the bail.	As expected.

Property Mortgaging Test - Check to make sure that opponents do not have to pay rent when landing on mortgaged properties.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
8	1. Start the program and begin a game. 2. Play the game for several turns and buy properties for each player. 3. Mortgage all of a certain player's properties through the mortgaging/unmortgaging menu. 4. Wait until the opponent player lands on a mortgaged property.	None.	A message should display informing the opponent player that they do not have to pay rent since the property is mortgaged.	As expected

Property Trading Test - Check to make sure that players can successfully traded properties.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
9	1. Start the program and begin a game. 2. Play the game for several turns and buy properties for each player. 3. Navigate to the trading menu as any player and select a player to trade with. 4. Trade several properties between both players. 5. Head back to the trading menu and select the same player to trade with.	None.	Each player should now have the properties they selected to trade with each other in the previous trade.	As expected

Declare Bankruptcy Test - Check to make sure that a player can declare bankruptcy and leave the game.

Test ID	Test Steps	Test Input	Expected Results	Actual Results
10	1. Start the program and begin a game. 2. Play the game for several turns and buy properties for each player. 3. Navigate to the Player Manager menu as any player and declare bankruptcy.	None.	The player should be removed from the game and all of their properties should become unowned.	The game freezes and an out of bounds error occurs.

14. Evidence of Bugs and Fixes for Bugs

14.1 Go Space Loop

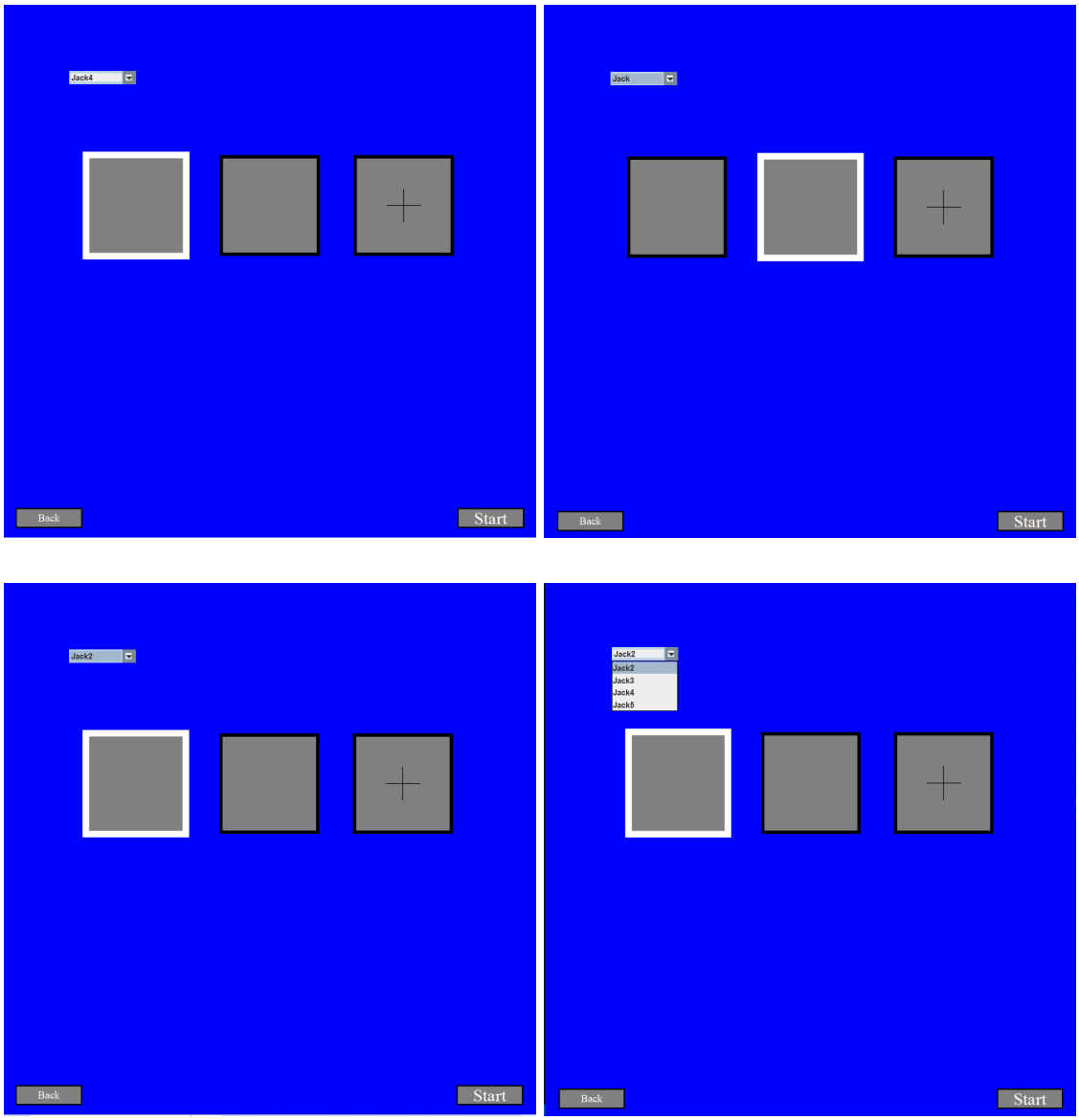
In previous versions of the code, there is a bug where if a player rolls a number that lands them directly on the Go space, they will be stuck in an infinite loop of the Go space message being displayed. There is a screenshot of this message below.

The reason this was occurring was due to the game switching to the PLAYER_MOVE_STATE state when the “Ok” button was clicked, as the player may still have spaces left to move if they rolled a number that landed them past the Go space. To fix this error, several changes were made to the code. In the MouseHandler class, which handles mouse input for the game, if the player is currently on the Go space, and has no more spaces left to move (meaning they have landed directly on the Go space), the game state will be changed to NEXT_TURN_STATE once the

"Ok" button is clicked, so the player may end their turn instead of being stuck in an infinite loop of the Go message displaying.

14.2 Profile Name Selection Bug

In the early stages of development for the create game screen, there was an error where once the user clicks off a player box, if the player reclicks the player box, the selected name for the player box is changed to the first available option in the profile names drop down list. An example can be seen through the screenshots below (the previously selected name “Jack4” is changed to “Jack2” when the same player box is clicked again, due to “Jack2” being the first available name in the profile names drop down list).



The reason this was occurring was due to how the drop down list component would assign its selected item to the first item in the drop down list whenever any player box is clicked. To fix this bug, a secondary profile name array was created to store all currently selected names so they can be assigned as the selected value of the drop down list for each respective player box.

14.3 Utility Rent Bug

In Monopoly, a utility property's rent is based on the number rolled by the player landing on the property, and the number of utility properties owned by a single player. If a player owns one utility, its rent is four times the number rolled on the dice. If a player owns both utilities, its rent is ten times the number rolled on the dice. In an early version of the code however, the rent was $0 for each utility. There is a screenshot of this occurring below:

The reason this bug was occurring was due how to the game handled player movement logic. When a player rolls the dice, the game calculated how many spaces the player had left to move by decrementing from the Dice class's result variable.. However, this would mean that once a player reaches their destination space (in this case, the utility property), the dice result would now be zero, meaning the rent of the utility property would be zero (since the rent for utilities relies on the result of the dice). To fix this error, a separate variable was created in the Player class named spacesLeftToMove, and is now decremented from instead of the result variable in the Dice class so the Dice's result variable can remain the same value throughout a player's turn.

15. Installation Guide

To run the game on a computer (Windows):

- First download the code from the game's GitHub repository

  (Repository Link: https://github.com/jackloague1/Java-Monopoly-Project).

- Ensure that Java is installed on your computer.

  (Download Link: https://www.oracle.com/java/technologies/downloads/).

- Locate the directory of where the repository was downloaded.

- Inside the repository folder, run the "Game.jar" file by right-clicking the file, selecting "Open with…" and choosing "Java Platform SE Binary." Alternatively, if the file is already set to open using Java, the file can simply be double-clicked to be run.

16. Recovery and Backup

The code for the game will be pushed to GitHub so there is always a download for it online. However, in the case that the GitHub page or website is down, the most updated version of the game will also always be on a personal laptop, ensuring that the program is safe even if it cannot be accessed online.

17. Service Reliability Engineering (SRE)

17.1 Service-Level Objective (SLO)

Because this program does not use its own server or has any components that need to be maintained over time, the game should be available whenever the GitHub website and the game's GitHub repository page are available. Users may download the game and its source code from its GitHub repository at any time as long as the Github website and the game's GitHub page are online.

17.2 Service-Level Agreement (SLA)

Starting December 13, 2023, whenever new features or bug fixes are added to the game and pushed to its GitHub repository, the repository's README will be updated to inform users of what has been added or changed since the last push. It is hoped that this transparency can increase the availability of the game, as users may not be aware of the availability of new changes without documentation. If the README file is not updated whenever significant changes are made to the repository code, then the availability objectives of the game will not have been met.

17.3 Service-Level Indicator (SLI)

It is ensured that the game and its source code will always be available given that the GithHub website and the game's GitHub repository page are online. The repository's README file will also be updated whenever significant new changes like bug fixes or added features are pushed to the game's repository. To measure this availability, starting December 13, 2023, the number of

times the game has new changes to its repository, and the number of times the README file of the game is updated will both be tracked. The number of updates to the repository README file should be equal to or greater than the number of updates to the repository code. If the number of updates to the README file is less than the number of updates to the code, the service-level objective will not have been met.

18. User Manual

18.1 Main Menu

When opening the game, the user will be greeted with the main menu. Here, there are two options to choose from: Start, and Profiles. Assuming the user is beginning the program with no saved profiles, Profiles would be the best option to choose first.

18.2 Profiles Menu

In the Profiles menu, the user will be prompted to enter a profile name if they do not have any saved profiles. Users may enter any name as long as it does not exceed fifteen characters and is a unique name (not already used by an existing profile). Upon entering a profile name, it will be displayed in the profile names list. Users may add up to 25 profiles.

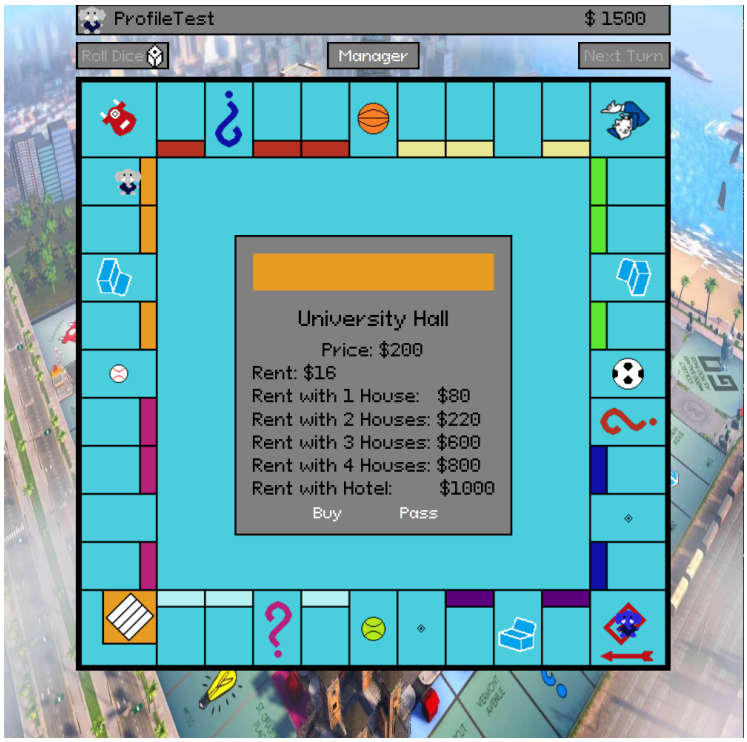
18.3 Game Set-Up Menu

Navigating back to the main menu, the Play option will navigate to the Game Set-Up Menu. Here, up to six players can be added to a game (however, as of current, there is a bug in the game where some buttons in the game lose their functionality by playing with four or more players). By clicking on any player square, a user can set the profile they would like to use through the drop down list on the left, and the token they would like to play as using the scrollable list on the right. Users may also create new profiles from this menu by clicking the Create Profile option at the center-bottom of the screen. Once every player square has a selected player profile and selected token, the game may be started by clicking the Start option in the bottom right corner.

18.4 Main Game Screen

Once the Start option is clicked in the Set-Up screen, the main game board will be displayed. The token, name, and money amount of the player whose turn it currently is will be displayed at the top of the screen. A user may begin their turn by selecting the Roll button.

As players roll the dice and move around the board, they will be prompted with certain instructions depending on the space they land on. Below, a player is prompted whether they want to buy or pass on a property.



When a player is finished with their turn, they may select the Next Turn button so the next player may begin their turn.

18.5 Player Manager Menu

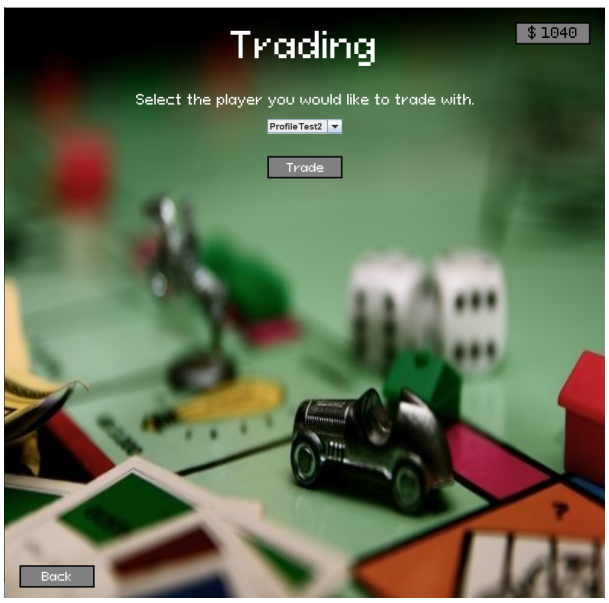
By clicking the Manager button located at the top-center of the main game screen, the user will be navigated to their manager menu. Here a player can perform certain actions, such as mortgaging and unmortgaging properties, trading properties with other players, and declaring bankruptcy if they cannot pay a debt or they wish to leave the game (the Build/Sell option unfortunately does not exist).

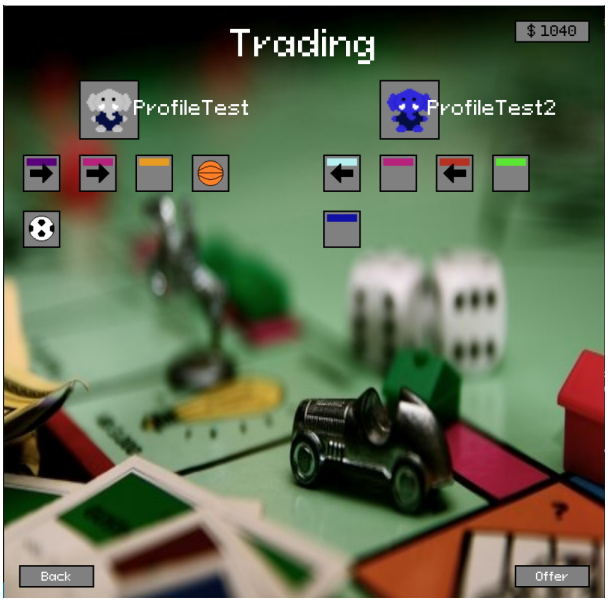
18.6 Mortgaging and Unmortgaging Menu

By selecting the Mortgage/Unmortgage option in the Player Manager menu, the user will be navigated to the mortgaging/unmortgaging interface. Here, all of their properties will be displayed, and the user may select any one of them at a time. By selecting a property, the user will be given the choice to either mortgage the property and receive money if it is currently unmortgaged, or unmortgage the property by paying a fee if it is currently mortgaged. Mortgaging properties can be useful if a player needs money to pay off a debt, while unmortgaging can be useful if a player wishes for an opponent to pay rent when they land on the former's properties.

18.7 Trading Menu

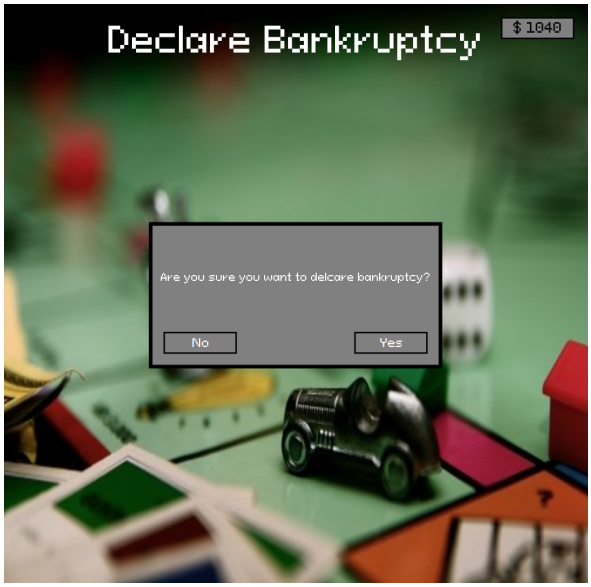
By selecting the Trading option in the Player Manager menu, the user will be navigated to the game's trading interface. First, an opponent must be selected from a dropdown list to trade with. Once they are selected, all of the properties of both the user and the selected opponent will be displayed. The user may choose any properties they wish to give away or receive by clicking on them. Once the user is satisfied, they may click the Offer button in the bottom right corner. The player being traded with will then be prompted if they wish to accept the trade. If the trade is accepted, the properties selected will be switched between the two players.





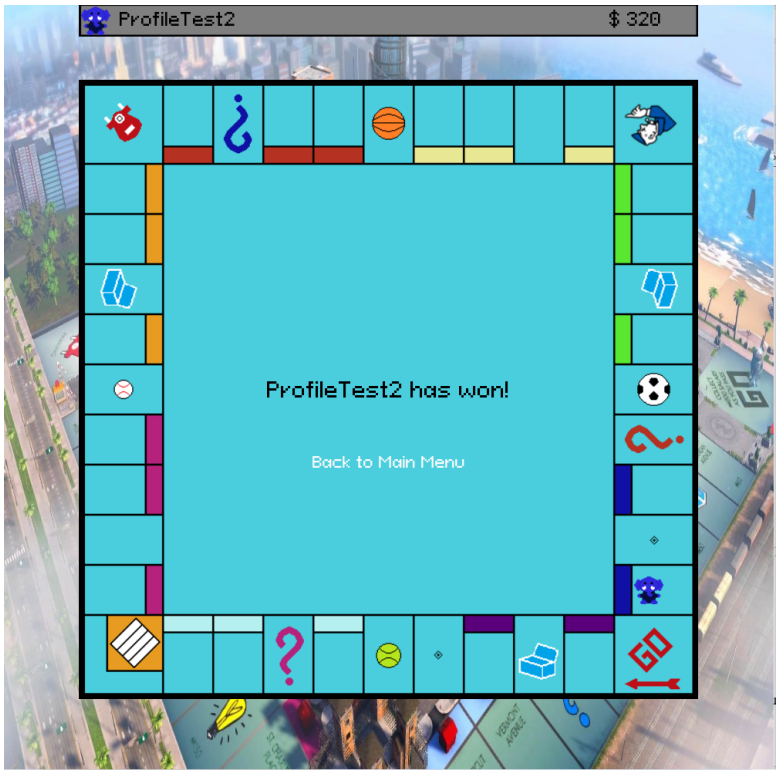
18.8 Declaring Bankruptcy

If a player does not have the funds to pay a debt or simply wishes to leave the game, they may choose the Declare Bankruptcy option in the Player Manager menu. Once they declare bankruptcy, they will be removed from the game, all of their properties will become unowned, allowing any remaining player to buy them.



18.9 Game Over

Once there is only one player left in the game, they will be declared the winner, and the game may be returned to the main menu by selecting the Back to Main Menu button.



19. Conclusion

This project aimed to bring knowledge from the various programming and concepts and methodologies learned over the Computer Science degree. The Scrum methodology was used to schedule the work for this project, with documents such as product backlogs and sprint backlogs defining features, user stories, and what tasks to assign for each week of the project's development. As for the programming side of things, this project provided valuable experience in the Java programming language and was a great refresher in object-oriented programming.

Overall, the goal of this capstone project was to create something GUI based using object-oriented programming, and recreating the video game Monopoly was a great exercise in obtaining valuable experience in both. This project also provided more insight into how the architecture of video games work, such as by using a main game loop that updates and renders all game objects within the program. While not all features could be implemented for the game, and there are quite a few bugs in the program, this capstone project provided great experience in documentation, scheduling, and programming.

References

Chia, D. (2021, September 16). *The Observer Design Pattern*. Medium. https://medium.com/@davidcwh/the-observer-design-pattern-891358282f31

Oracle. (n.d.). *Package java.awt*. https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html

Oracle. (n.d.). *Package java.swing*. https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html