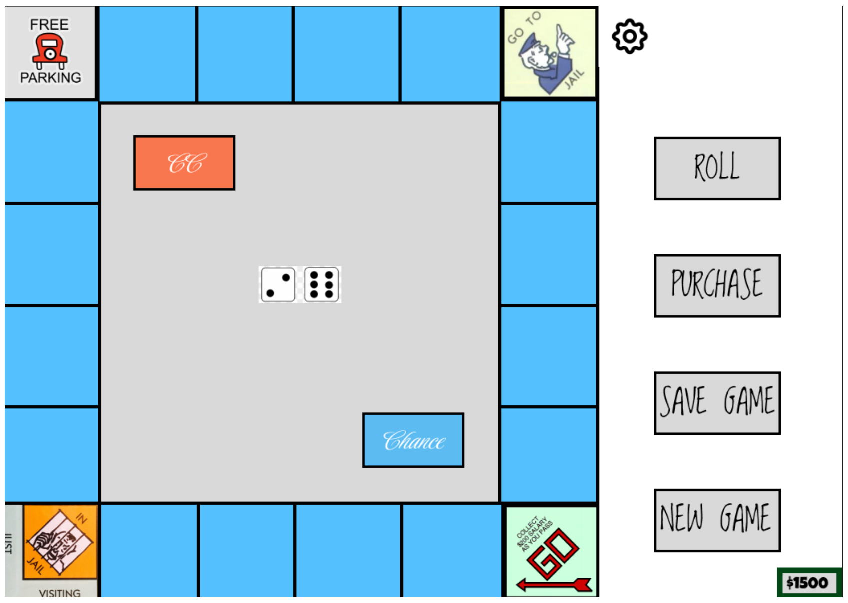
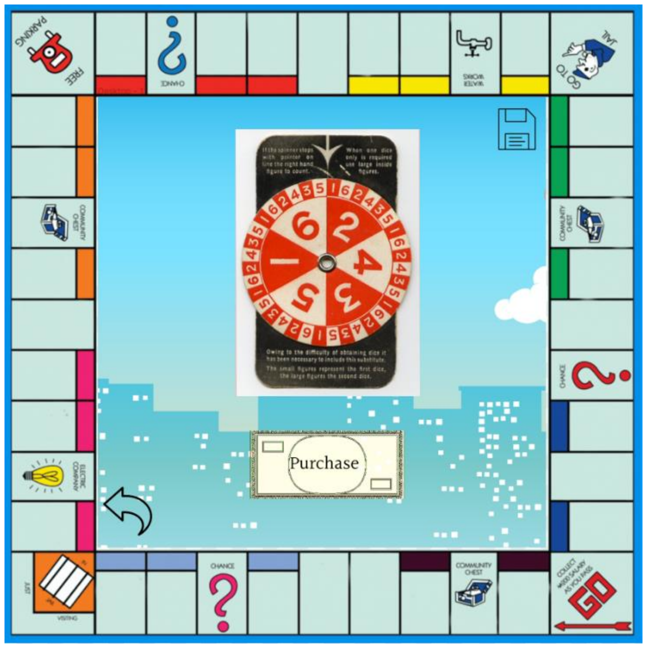
UI Prototyping Document

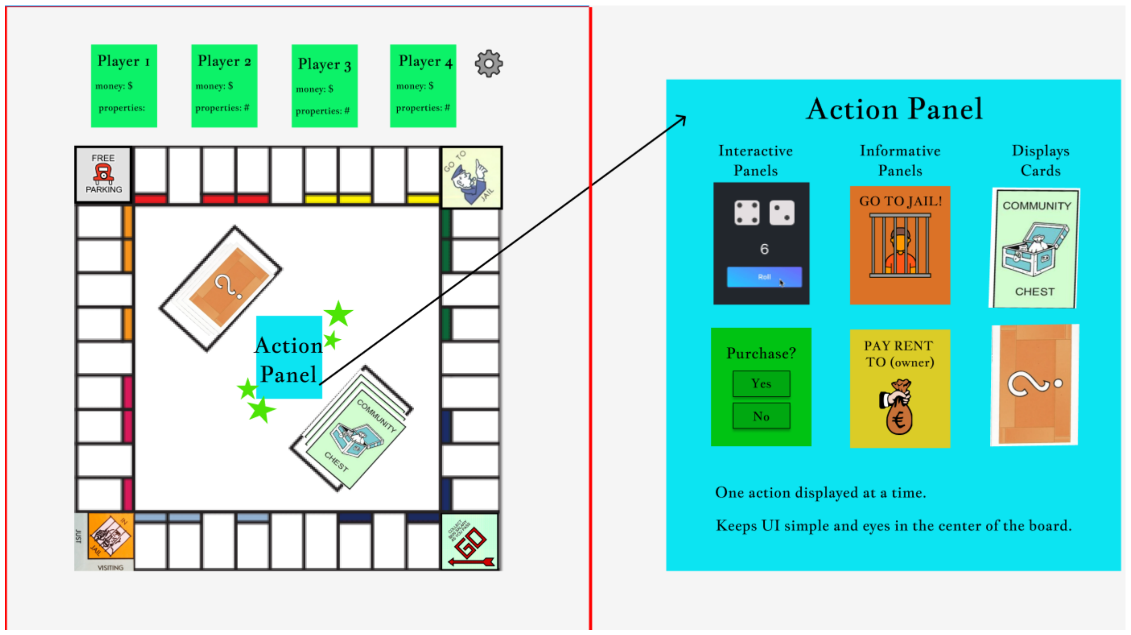
Gameplay Design References:





A

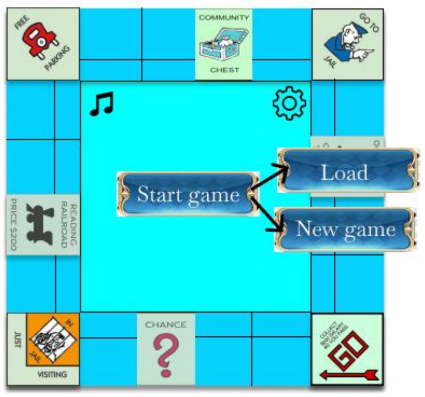
B



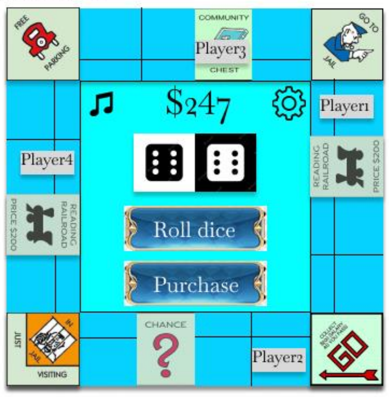
C

D & E

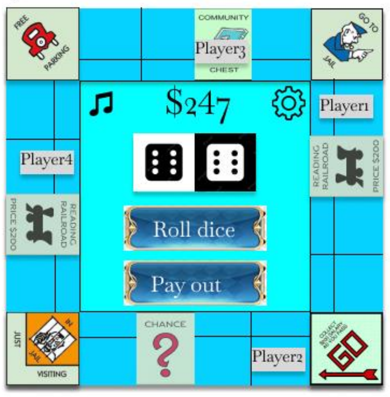
Main menu



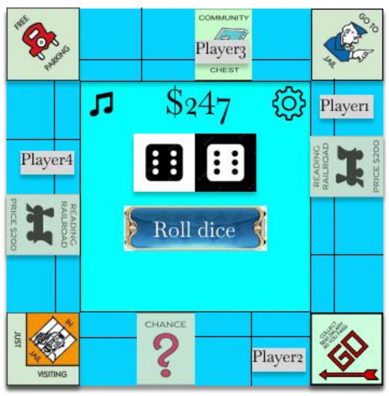
If on the city card



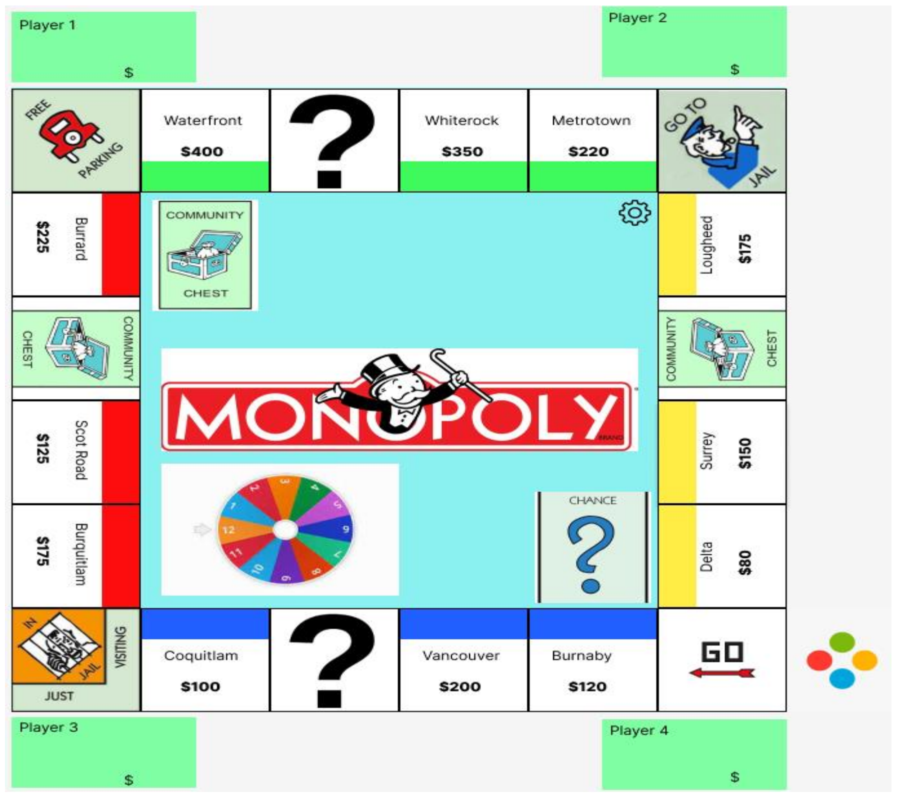
If in jail



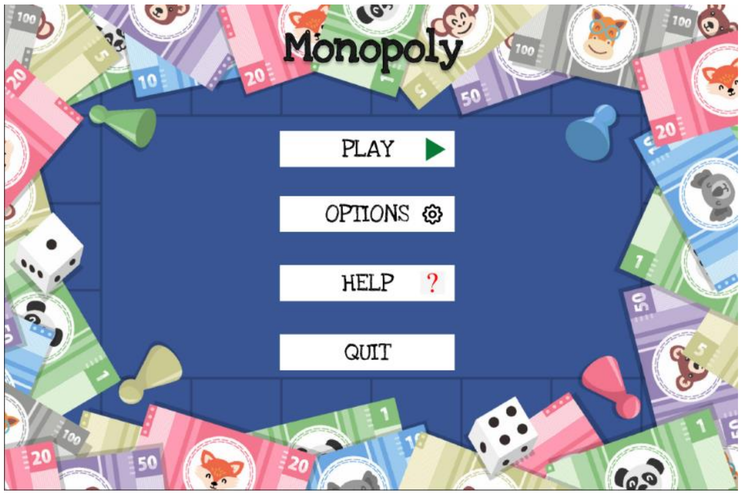
If on parking, chest, chance, start, reading railroad



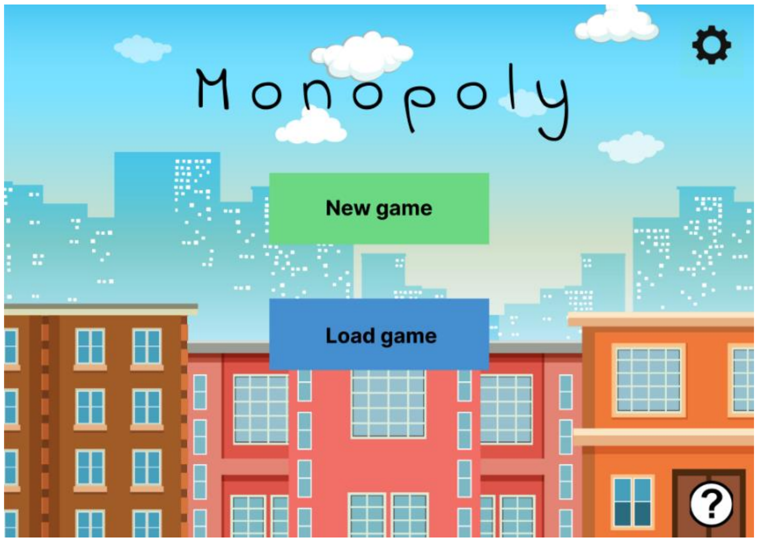
F



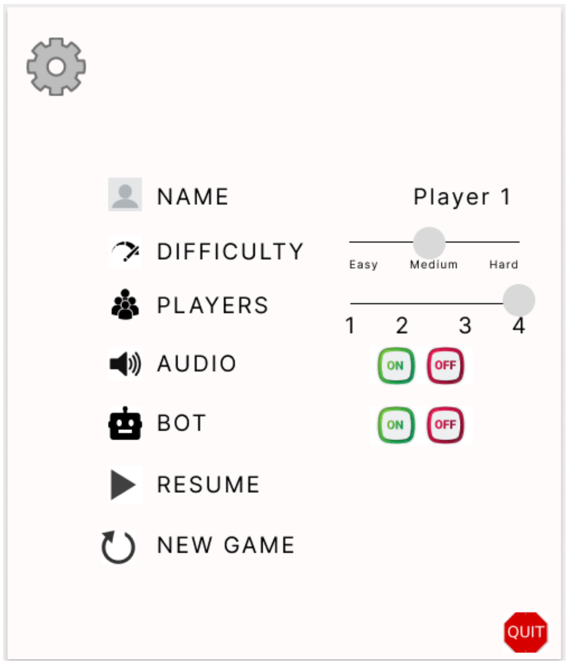
Other Prototype References:



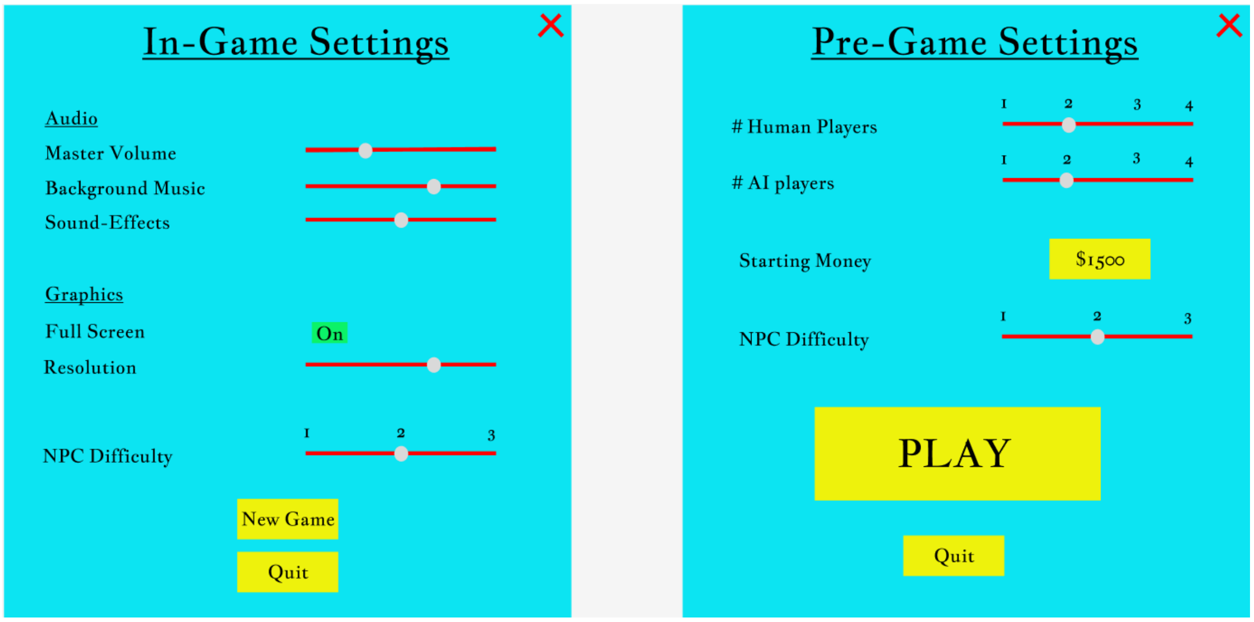
G



H



I



J

UI Prototyping Discussion:

Design A:

Strengths:

Design A achieves consistency by using the same font and size for all buttons. A simple and clear interface is achieved by separating the board display and the player accessible buttons. The user is given more control as all buttons are displayed on the screen at the same time, regardless of whether a button press is considered an illegal move in the context of the game. In addition, the settings cog would lead to a menu which allows the player to adjust properties of the interface, such as audio options and screen resolution options. The presence of the simple buttons which are always displayed on the screen and the presence of text also helps the user to understand how to use the interface better. While not flashy, it helps to avoid confusion. Finally, recognition is used in this design over recall with the presence of the buttons. The player does not need to memorize how to perform a specific action such as moving or purchasing, and the text within the buttons also helps to remind the player what its specific function is.

Weaknesses:

The state of the game is not communicated very clearly to the player in this design. The number of players is not displayed, nor the properties that they own. In addition, the balance of the current player is displayed in a small corner in the bottom right, which is unintuitive and can be easily missed. This violates the UI principle involving focus and attention. In addition, colour theory is violated for the text displayed on the Chance and Community Chest cards. The white text does not stand out against the backgrounds very well and may cause difficulties for visually impaired players.

Design B:

Strengths:

Design B does a good job of helping the user to focus their attention. While in design A the buttons were essentially separate from the board, forcing the player to focus on 2 different parts of the interface, in design B the buttons (such as Purchase, Save, Return) are directly on the board. This helps the player to focus on one portion of the interface at a time.

Weaknesses:

Design B does not communicate the state of the game very well. The number of players, current balance, and owned properties are not displayed. In addition, recognition could be improved in this design as the save and return buttons do not have any text to accompany them. The player may not know the purpose of the buttons just by looking at them and some sort of text should be provided to avoid confusion. Finally, the contrast between the background and the buttons is weak. The buttons are hollow and do not particularly stand out, which may cause issues for visually impaired players.

Design C:

Strengths:

Design C does a good job of communicating the state, displaying the current number of players as well as the stats of each player. In addition, the action panel in the middle of the screen will check the current game state and display the corresponding information. For example, if a player has landed on an unowned property, the option to purchase the property will appear. This design also helps the user by only displaying the buttons that are relevant at the time. This helps prevent the user from potentially making an illegal move, such as attempting to purchase a property when they have landed on an unpurchasable square. In this case, the option to purchase would not appear and potential errors would be prevented. Finally, this design helps the user to focus by having the action panel be displayed in the middle of the board, while only keeping relevant buttons on the screen as to not create clutter and confuse the player.

Weaknesses:

The main weakness of this design has to do with consistency. The panels to be displayed all use different fonts and font sizes. If multiple panels are displayed at the same time, the result could be messy and visually displeasing to the player. For example, if a player owns all cities in a set and is able to purchase houses, the panel to purchase a house would need to be displayed, alongside whichever other panel may be displayed at the time, such as the dice panel.

Design D/E:

Strengths:

Similar to design C, this design helps the user by only displaying the buttons that are relevant to the player's current situation. In exchange for user freedom, this design favors simplicity and error prevention. This design also achieves consistency by maintaining essentially the same interface between the main menu and the gameplay. The buttons are of the same style and the board to be used in gameplay is also displayed in the background. Finally, this design makes use of analogous colours for the UI, with three shades of blue. This is an example of good colour theory.

Weaknesses:

This design may have an increased cognitive load on the player. There is a lot of information displayed in the middle of the board at the same time, and not many labels to provide the player with information. It may be unclear to the player whether the money displayed is the player's balance, the price of a property, or the rent that needs to be paid. In addition, the space is quite cramped and could be overwhelming if more information needs to be displayed, such as the display for a chance or community chest card.

Design F:

Strengths:

Similar to some of the other designs, this design helps the player to focus on the centre of the screen as all relevant buttons and displays exist there. This eliminates the possibility of a player being unable to find something, because everything is focused on the middle. For example, in design C, the settings cog is small and placed at the very top right corner of the screen, which may cause players to miss it entirely. In addition, this design does a fair job of communicating the player states, as each player and their corresponding balance is shown on the screen at all times.

Weaknesses:

This design lacks any text and this does not help the player. Players may be unaware that they need to click on the wheel in order to make a move/roll the dice. In addition, the graphics on the board are inconsistent in size, which may appear messy to the player.

Design G:

Strengths:

This design does a good job of being consistent. Each button is the same size with the same background colour, and the same font is used. The design is quite minimalistic and aesthetic, as no unnecessary information is displayed. In addition, the black text on white background creates good contrast and is an example of good accessibility. Colour is not important in this design and there are symbols beside the buttons in order to further assist the player.

Weaknesses:

The background in this design is quite messy and cluttered. This could diverge the player's attention away from the important part of the interface, the buttons.

Design H:

Strengths:

This design does a good job of presenting a logical visual hierarchy. The options to play the game (new game & load game) are displayed right in the middle of the screen, as that is what the player ideally wants to do. The settings and help icons are smaller and placed away from the centre. In addition, this design provides a consistent theme with the gameplay UI in design B, utilizing the same background image and giving the player a feeling of consistency.

Weaknesses:

The buttons on screen could be made to contrast better with the background by adding a dark border or similar, as they are currently similar in shade, which could cause issues for visually impaired players. This also causes the buttons to stand out less. Finally, there is a large inconsistency between the title text and the button text. The fonts are drastically different in both style and features. The title “Monopoly” text feels similar to Comic Sans, while the Inter font used for the buttons is much more serious. The two font styles clash with each other and do not particularly fit because of their stark differences.

Design I:

Strengths:

This design for a settings menu is very accessibility friendly. The design makes use of symbols, text, and other features such as sliders to ensure the player understands what is going on. The black and white design is simple and clean, and makes colour an unimportant part of the design, which could benefit those who are visually impaired. The font and symbols are also consistent, with the same spacings in between each. The symbols make sense and do a good job of supporting the text.

Weaknesses:

This design can be slightly confusing, as the difficulty setting is unclear in what it does. The player will be unsure whether the difficulty refers to the difficulty of the AI (if the AI exists), the difficulty of the game (starting money), or something else entirely. It would be greatly beneficial if there were small tooltips that could be displayed on hover in order to explain the functionality of each setting.

Design J:

Strengths:

This design is consistent in the font used and font sizes. The user is given more control in comparison to design I by allowing customization of various audio settings and using sliders for everything rather than simply having audio on or off like in design I. In addition, the settings are communicated clearly and is unambiguous in the language, which helps to avoid confusion. For example, the NPC difficulty setting is more clear than the difficulty setting in design I and will not be mistaken for something else.

Weaknesses:

This design has a visually unappealing color scheme and is not a proper utilization of triadic colours. In addition, there are no symbols or textures as visuals to aid the text, which could cause confusion for players. There may be players who are unfamiliar with what an “AI” is and a robot symbol or similar could help to further clarify and provide information to the user. There is also slight inconsistency, particularly between the “# AI Players” setting and “NPC Difficulty”. The player may not understand that AI and NPC refer to the same thing, and may be confused by the function of the NPC difficulty setting. Consistent wording should be used to help avoid confusion. Finally, the text does not stand out very strongly against the blue background. In the In-Game Settings, headers such as Audio and Graphics could be bolded or made larger in order to show a proper visual hierarchy. The lack of boldness and contrast could also cause issues for visually impaired players.

Design Patterns:

1. **Factory method:**
   We implemented the factory method in our code to handle the squares on the Monopoly board. There are three different types of squares on the board: city squares, utility squares, action squares (chance & community chest), and other squares (free parking, go to jail, etc.). Each type of square would have a different functionality and variables when interacting with the player. Therefore, the factory method was intuitive in order to produce each type of square, which is why we chose to implement it. First, an abstract base class (Space.java) was created, with a Boolean field isProperty, a name field, and an action method (to be overridden by the subclasses). Each subclass implements the action method differently: the City (property) class checks whether a property has already been purchased, whether houses are built, and determines who rent needs to be paid to. The Utility class is similar to the City class but does not allow hosues to be built on the property. In addition, there is an extra multiplier field which helps to determine the rent that must be paid. The ActionCard (chance & community chest) class generates a random action for the player, such as receiving money or receiving a get out of jail card. Finally, the OtherSpace class handles the rest of the squares, depending on the position of the player. This design pattern improves our codebase because it provides an easy way to determine what action to take when a player lands on a certain space and eliminates the need to use a long string of if/else if statements or a switch statement. Instead, the action() method will simply be called and the corresponding output will happen based on what type of Space the current square is. Without this pattern, the game would need to find a way to determine whether a space on the board was a property or another type. Using this pattern, all that needs to be called is the action() method, which is implemented by each subclass.
2. **Observer:**
   We implemented the observer design pattern in order to provide a seamless and efficient way to decide which buttons to display on the screen through simple communication between the frontend and backend code. The PlayerObserver interface describes two methods that will be called when the player’s state changes and when a game over occurs. Any class that wants to be notified of these events will implement the interface

and subscribe itself to all players. For example, whenever a player rolls the dice and makes a move on the board, the position is updated in the Player class and notifies all of its observers by iterating through its subscribers and calling their onPlayerState method. The implementation of onPlayerState in each observer, will determine how the GUI will updated. If the player has landed on a property, the observer would display a purchase option. We chose this design pattern because it is a good way to easily communicate between different classes while affecting the minimum amount of code; it is a great example of open for extension closed for modification since we can extend the functionality of an existing class by making it implement the PlayerObserver interface without modifying existing code and keeping the Player class untouched. This helps us keep track of the game state and manage which UI elements should be displayed at which time. While there are only a few observers implemented for the current prototype, this design pattern will greatly improve our codebase later on when more GUI elements are added. Without this pattern, checks would need to be implemented in each class that may affect the game state and more variables would need to be created. It would also lead to a large monolithic Player class where each GUI element would need to be a member of the Player class in order for us to update the GUI.

3. **Prototype:**

   We implemented the prototype design pattern to ensure a consistent UI design for the player statistics. The prototype combines all of the common elements of the menu into one object that can then be cloned. The PlayerStatMenu is an abstract class that contains all of the GUI elements that make up the player menu. PlayerStats extends this class to implement the abstract method, clone. Now we can clone the prototype and modify any existing parameters that are unique to an instance. For the current version of the game, the main differences in each PlayerStats object mostly comes down to their position. In the future, we can add more concrete classes that extend PlayerStatMenu to have unique displays for different types of players like AI. For example, the AI will have the same menu as real players, but may need a label displaying its current difficulty. This can be easily added by creating an AIPlayerStats class and adding the extra label without copying code. If the PlayerStatMenu UI design changes, both PlayerStats and AIPlayerStats would be updated and consistent with each other.