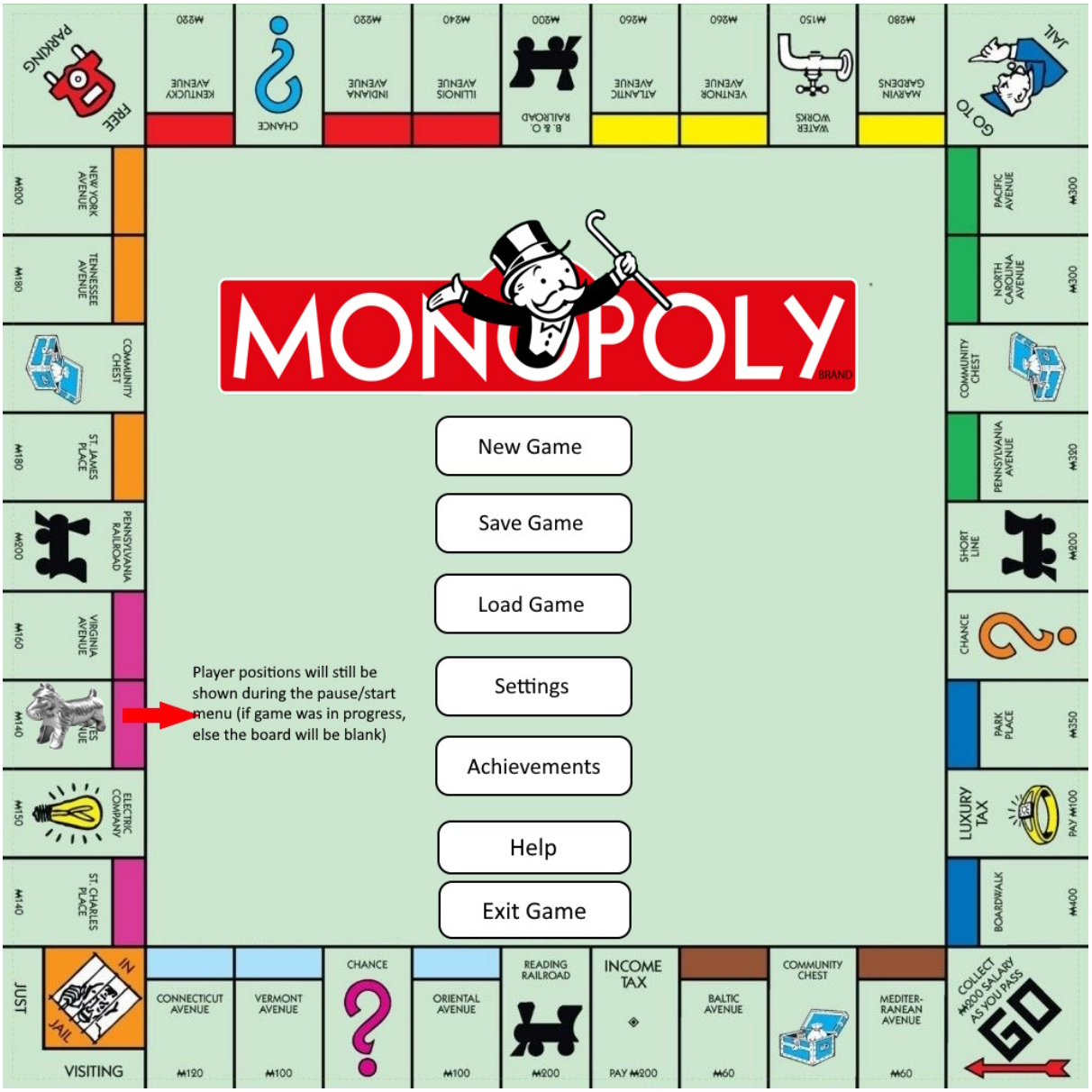
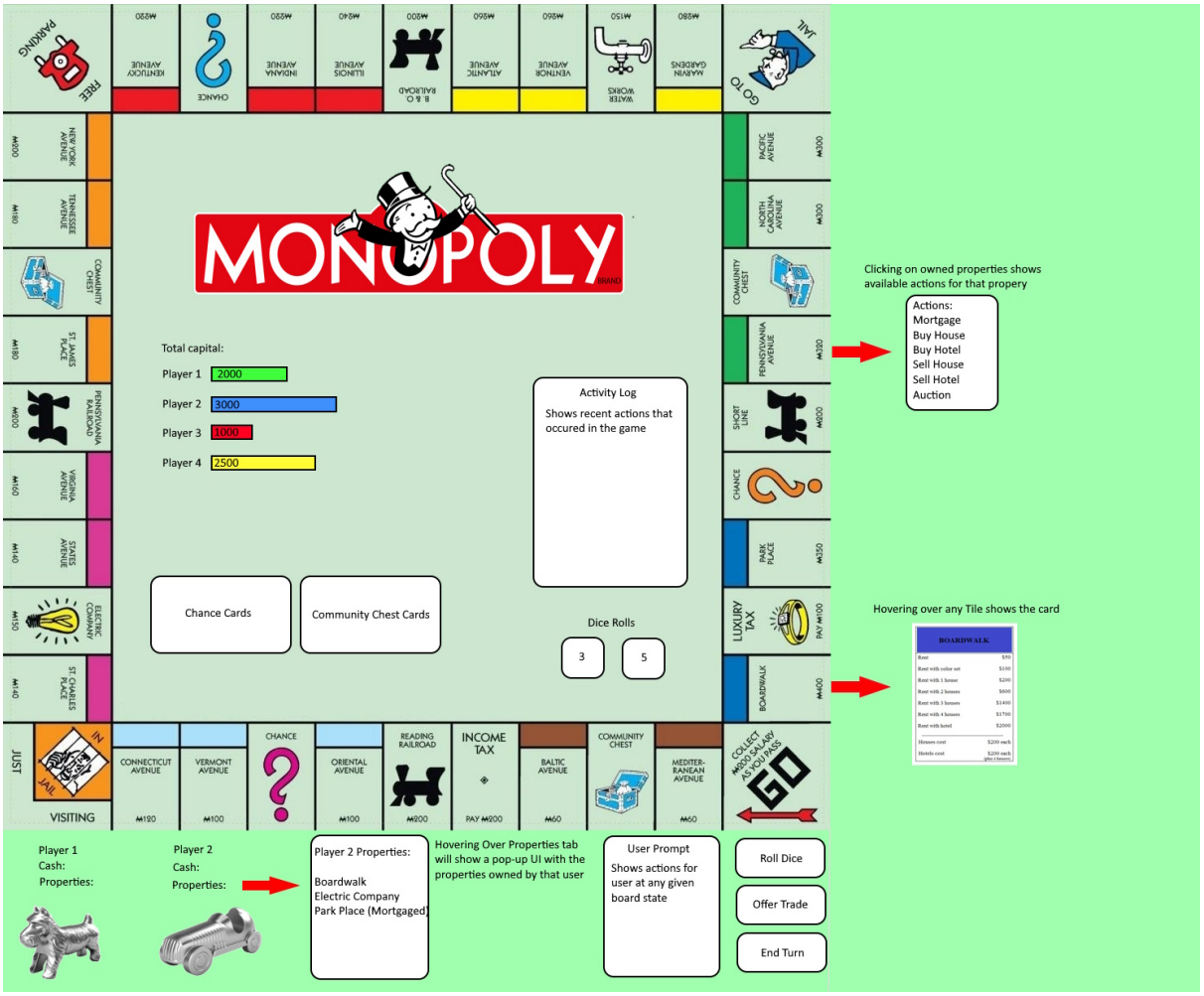
Discussion of final UI evolution

In the group discussions, we compared and analyzed all prototypes we prepared for the Monopoly game, taking into consideration the UI design principles, including the ten heuristics of the Nielsen Norman Group. Here is an analysis of the strengths and weaknesses of each prototype, and how the final UI evolved by incorporating the strong elements of each:

Prototype 1: Nick Chu





Strengths:

- Clear visibility of system status through informative notifications-notifies user about the current situation such as player's current properties, player's current position at the board, activity log, players' piece etc.
- The main UI is very appealing as it displays the buttons in the centre with monopoly logo at top. The user will find consistency and standard in the UI, will be able to figure out how the system works based upon previous experience with different interfaces.
- Match between system and the real world: The system is using language, concepts, and conventions familiar to the user, making information and actions appear in a natural and logical order.
- The players' balance is displayed in the centre would help each one to take decisions based upon their current financial situation, also the bar used to display the balance is

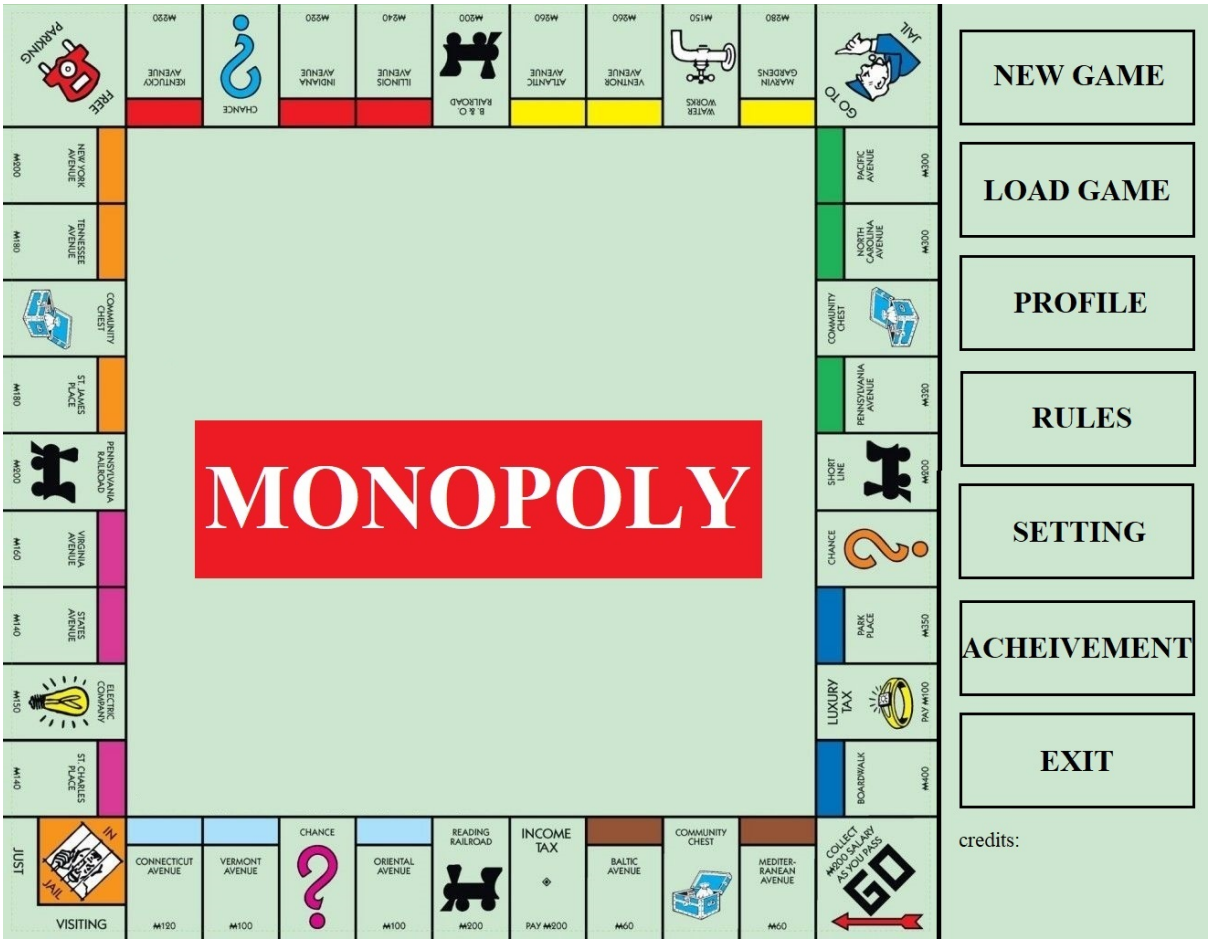
visually more helpful to compare the numbers with other players rather than looking at actual numbers. Reduces cognitive load on user.

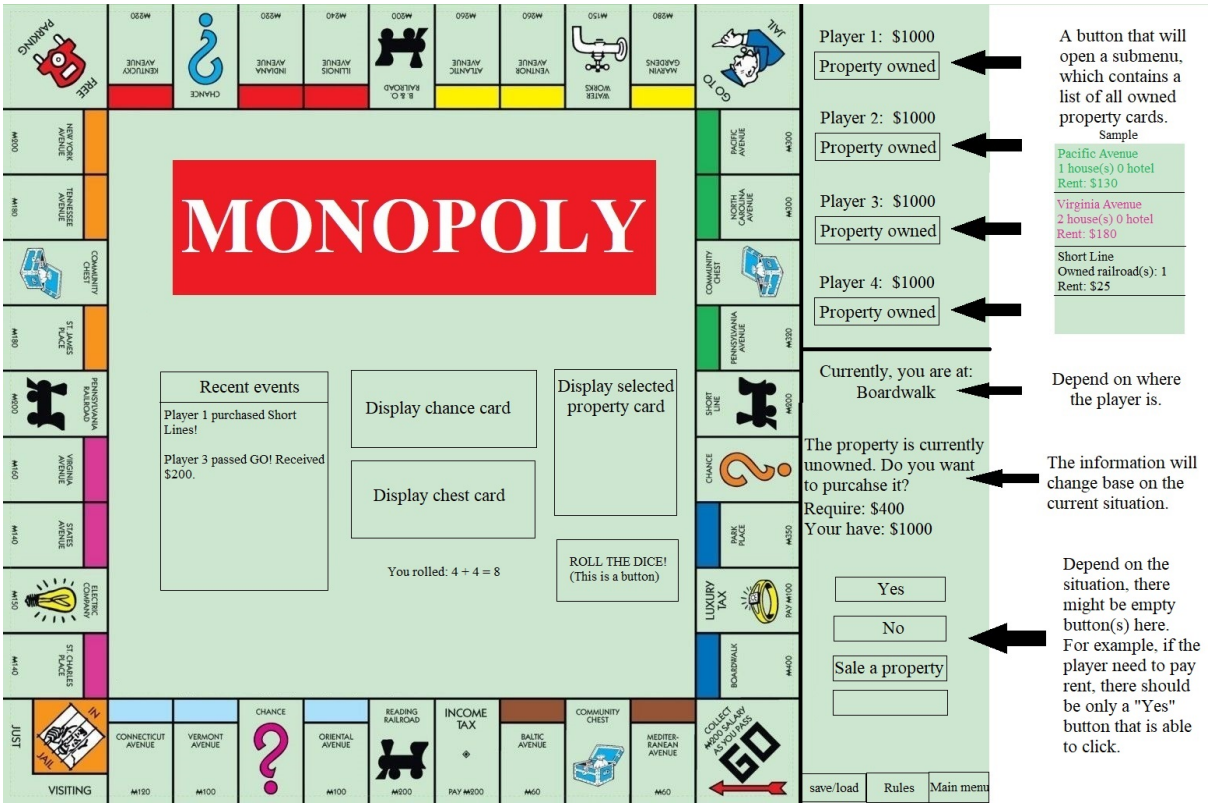
- Flexibility and efficiency of use: Expert players can navigate through the game quickly, while novice players can access additional help and guidance as needed.

Weakness:

- Lack of user control and freedom in certain interactions, such as no choice for saving the game and come back later.
- No option for the user to access the game rules and other main menu options during the play which affects the principle of recognition rather than recall as the user has to remember the rules etc.
- Help and documentation principle is not followed on the gameplay screen as there is no access to rules and assistance.
- Limited error prevention measures.

Prototype2: Henry Rao





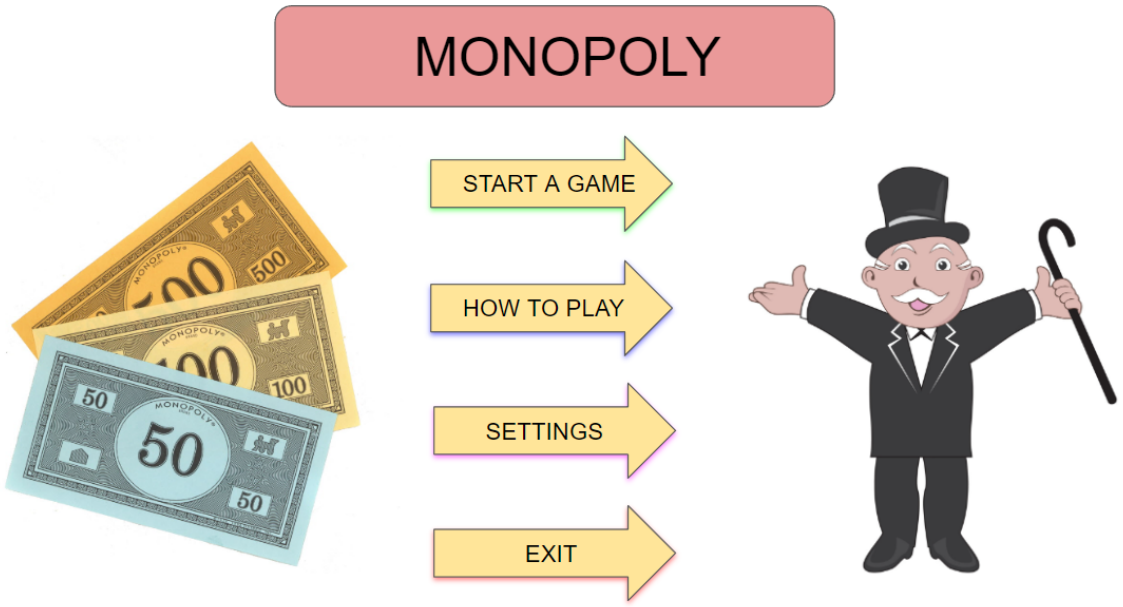
Strengths:

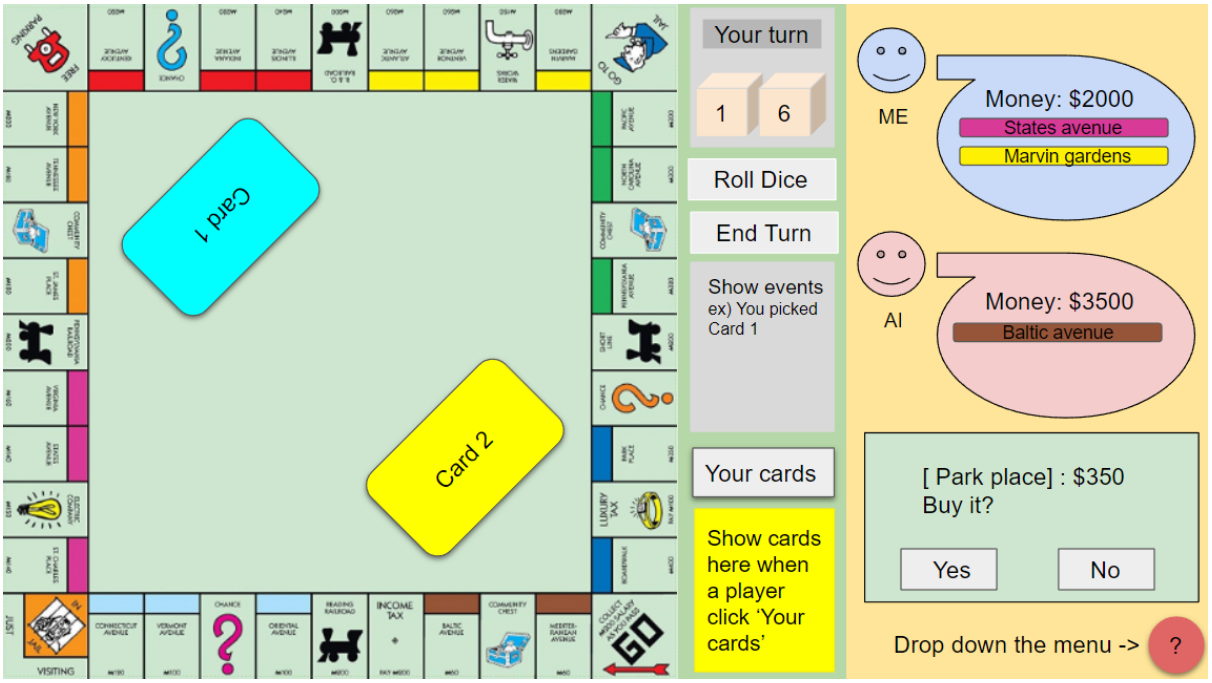
- Clear visibility of system status through informative notifications-notifies user about the current situation such as player's current properties, player's current position at the board, recent events, etc.
- Consistency in design, with familiar language and conventions. The language used is similar to the original Monopoly game and can be understood easily.
- User control and freedom with options for choice- User has the choice to buy properties, presents current balance of user and power to take decision based upon current financial status.
- Recognition rather than recall- the gameplay UI has rules button which allow the user to access the rules at any time rather than recalling it from the main menu. Also, the display of player's current properties helps the user to access the info at any time, instead of remembering what the player owns.
- Display of selected card enhance the visibility of the system.

Weakness:

- No option for auction the property, trade, mortgage/unmortgage which limits the flexibility and efficiency for expert users who would like to try for these options during the game.
- In the main menu, the buttons should be in the middle as the user enters the game environments focuses on the centre for information.

Prototype 3: Yewon Park





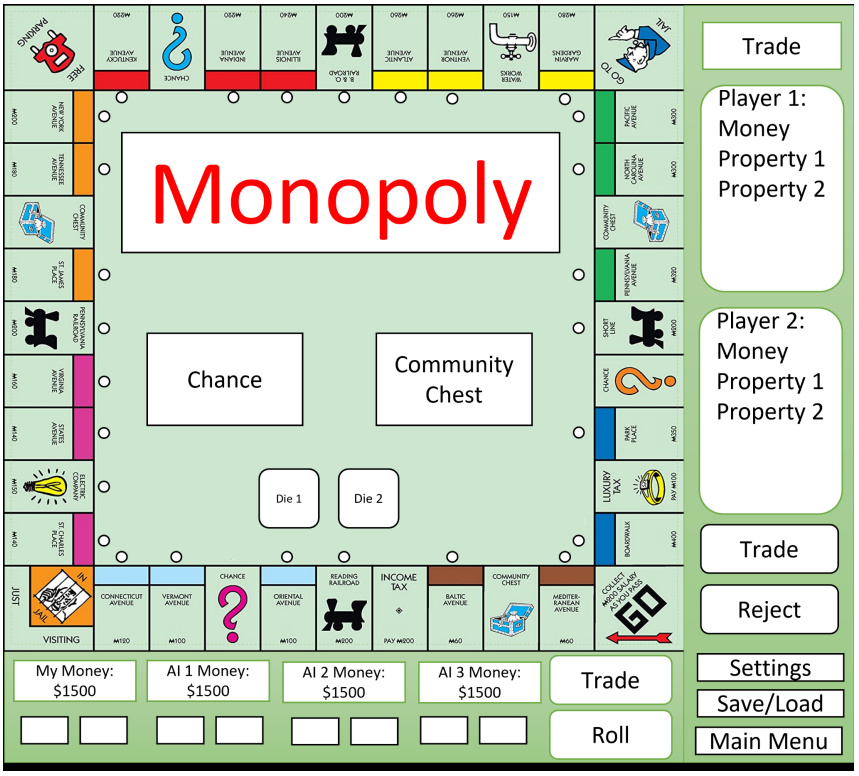
Strengths:

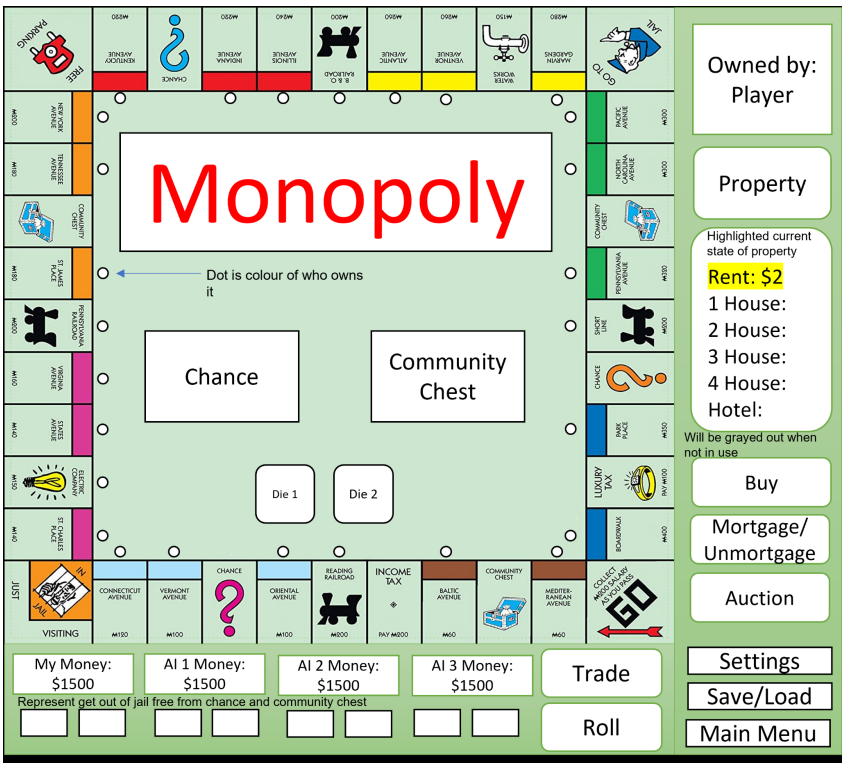
- Clear visibility of system status through informative notifications-notifies user about the current situation such as player's current properties, show events, notification for the choice to buy property.
- In the main menu, money display makes the user to predict the game that it involves money exchange which is helpful for the new user.
- Help and documentation: The system provides context-sensitive help and documentation, readily accessible from the interface such as the drop-down menu to access the settings and play rules.
- Match between system and the real world: the language used is easily understandable by the user and main menu is quite similar to the interfaces we encounter everyday, so easy to follow.
- In the game play screen, the player information and other sources are designed on one side which is helpful for player to access everything at one corner rather than roaming around the screen.
- Match between system and the real world: The UI incorporates familiar language, concepts, and conventions from the traditional Monopoly game, making it easier for players to understand and navigate.

Weakness:

- Limited user control and freedom: The players do not have choice to mortgage their properties to stay in the game.
- Lack of Consistency: The main menu design is not consistent with the gameplay screen which seems odd. The monopoly logo in the main menu is not present in the gameplay screen.
- Insufficient error prevention measures.
- The main menu buttons should not be arrows as the arrow symbol means it's pointing to something, however in this case the button itself needs to be pressed to display the required information such as 'how to play.'
- Lack of help and documentation- the 'how to play' button should be present in the gameplay screen rather than accessing it from the main menu.

Prototype 4: Stephen Ehebald





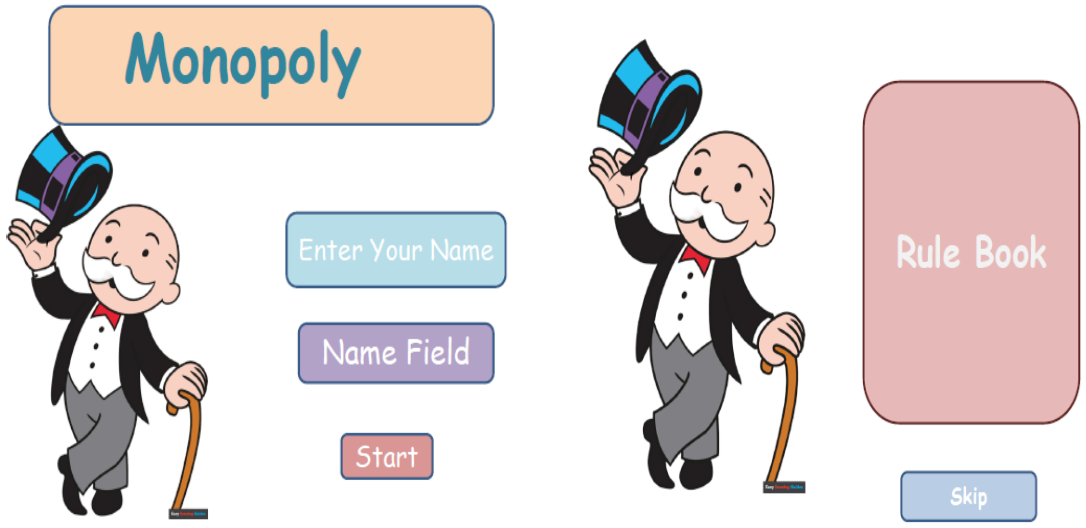
Strengths:

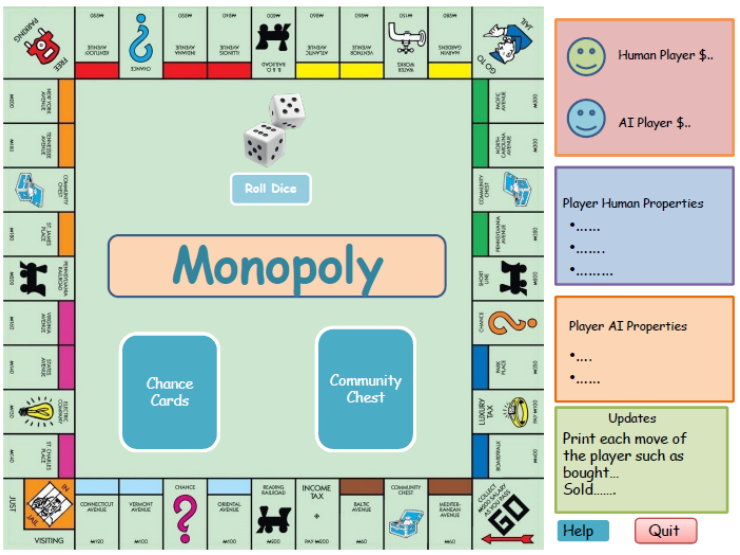
- User control and freedom: the UI provides control and freedom to the user as the player has access to choices such as trade, reject, mortgage/unmortgage player property information control which helps the user to take decisions, options to exit the game. Includes the current state of the property.
- Consistency and standards: The UI follow consistent design patterns, ensuring that visual elements, language, and interactions remain consistent throughout the game. Established conventions from the Monopoly game are incorporated, maintaining familiarity for players.
- Recognition rather than recall: Information, options, and actions are presented in a visible and easily accessible manner, reducing the need for players to rely on memory or recall. Game-related information, such as player details, and property details is readily available during gameplay.
- Match between system and the real world: The UI incorporates familiar language, concepts, and conventions from the traditional Monopoly game, making it easier for players to understand and navigate.

Weakness:

- Trade button is placed three times which is extra information and should be eliminated to aim for aesthetic and minimalist design.
- Limited visibility of system status as there is no display for the events occurring on each roll, this may confuse the player for further actions.
- The monopoly logo in the screen does not seem appealing, should have bold font and be creative one as per the game environment.
- Lack of comprehensive help and documentation- No access to game rules from the game play screen.
- The property information and the player's money balance information should be close to each other (should be on one side) as this makes player to access all relevant information at one place while playing. This will have less cognitive pressure on the player.

Prototype 5: Sumandeep Kaur





Strengths:

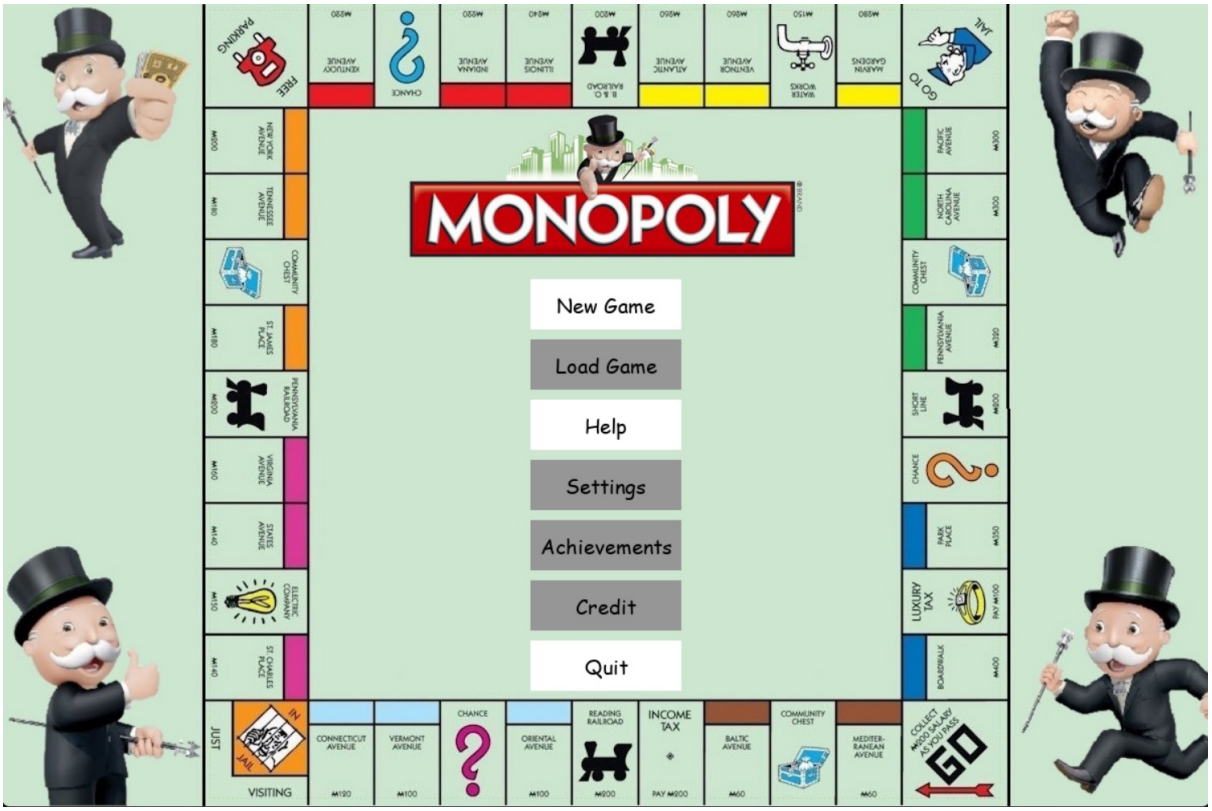
- Visibility of system status: Clear and informative notifications provide users with timely feedback on the current state of the game. Players are constantly aware of their turn, game progress, and any relevant updates.
- Match between system and the real world: The UI incorporates familiar language, concepts, and conventions from the traditional Monopoly game, making it easier for players to understand and navigate.

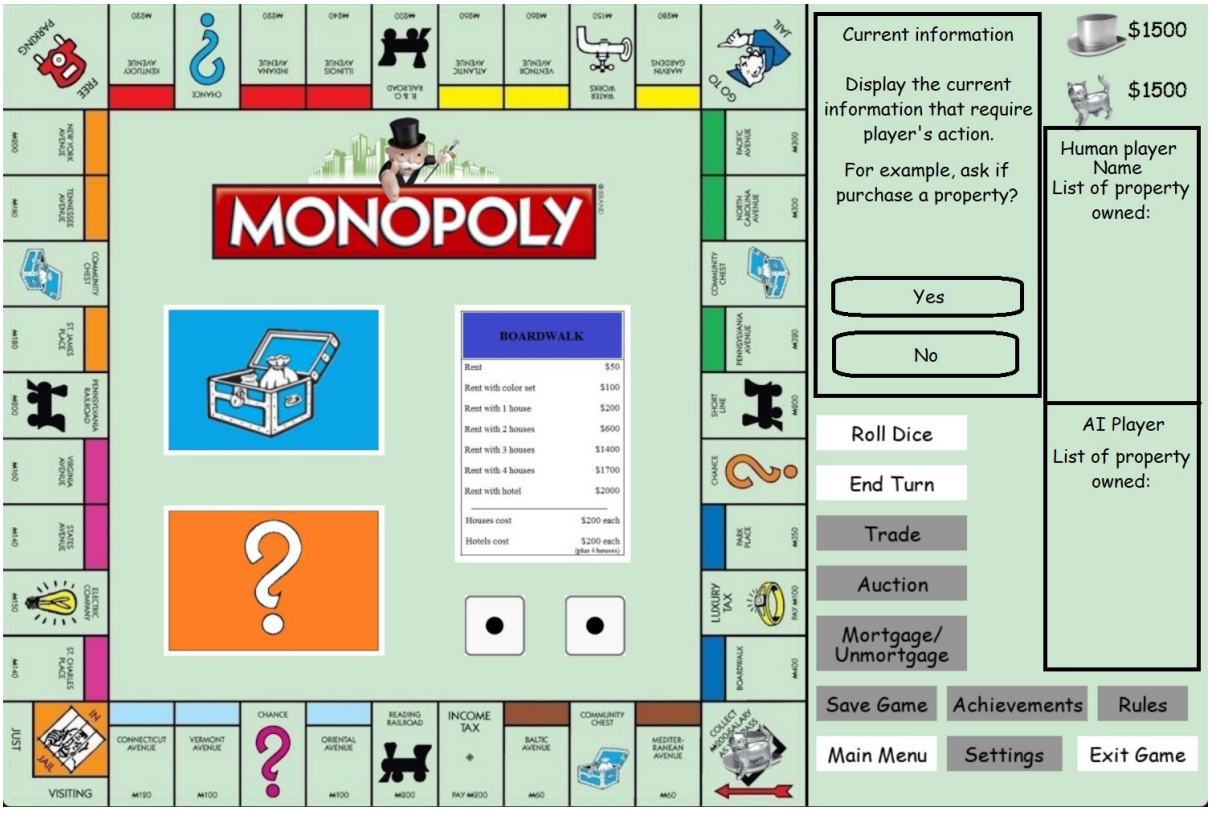
- Recognition rather than recall: Game-related information, such as player details, property details, and current game state, is readily available during gameplay.
- The main screen includes the option for the player to enter name, this provides the player a sense of belonging and identity in the game.
- In the game play screen, all the game buttons and information are on one side which helps the player to access information easily.
- Recognition than recall: the property information, money balance information, updates help the player to keep track of the game rather than recalling everything in mind, reduce cognitive pressure.

Weakness:

- Lack of comprehensive help and documentation- help button and other assistance is not present in the game play screen.
- User control and freedom: The UI does not provide players with control over their actions such as options to buy the property, trade, auction, mortgage, end turn, etc.
- Lack of flexibility and efficiency for expert players-As the main screen display rules for everyone, this may waste player's time each time the game is opened to play.
- Player's money and properties are not available together such as money is displayed at different place than property which affects the principle of aesthetic and minimalist design. Some of the information may be overlooked due to distraction.
- No options for saving and loading game, reduces game efficiency.

Final UI:


MONOPOLY
New Game
Load Game
Help
Settings
Achievements
Credit
Quit


MONOPOLY
BOARDWALK
Rent $50
Rent with color set $100
Rent with 1 house $200
Rent with 2 houses $600
Rent with 3 houses $1400
Rent with 4 houses $1700
Rent with hotel $2000
Houses cost $200 each
Hotels cost $200 each
Current information
Display the current information that require player's action.
For example, ask if purchase a property?
Yes
No
$1500
$1500
Human player Name
List of property owned:
AI Player
List of property owned:
Roll Dice
End Turn
Trade
Auction
Mortgage/ Unmortgage
Save Game
Achievements
Rules
Main Menu
Settings
Exit Game

Based on the analysis, the final UI for the Monopoly game evolved by incorporating the strong elements of each prototype while addressing their weaknesses:

From Prototype 1: We incorporated the main UI which has all the required information such as new game, load game, achievements, quit, rules, etc. This helps to satisfy the principle of help & documentation, and flexibility & efficiency. New players can have detailed information about each component before starting the game. We also included to idea of having player's pieces to play on the board, which helps the player to track their current position at the game board, satisfying visibility of system status heuristic.

From Prototype 2, 3, and 5: We decided to take the idea of displaying all information in the right side in the gameplay screen so that player can access all the information easily at one side, satisfying the heuristic aesthetic and minimalist design, focusing on essential elements, and reducing visual clutter. Moreover, to display the selected community or chest card is another idea to include from prototype 2 as this would enhance visibility of system status.

From Prototype 4: We got the idea to include all buttons such as auction, save game, load game, mortgage/unmortgage, trade in the game play screen. The buttons should be places at one place for easy access. We have decided to include all these buttons to match the traditional monopoly game so that consistency and standard principle can be followed, and user can predict the game from previous interface experiences.

From Prototype 5: We included the idea of giving player a choice to have a name in the game, this would help player to recognize itself in the game and avoid confusion with other players.

Following ideas have been incorporated from all prototypes:

- All prototypes followed the principle of 'Match between system and the real world' that most of the users will be familiar with language and concepts, making information and action appear in a natural order. We have included the idea to print event information on the screen as most of our prototypes have it. This step is quite useful for the player to track the status of the game and hence satisfying the principle of 'Visibility of System Status'.

- The game Buttons are included to provide user control and freedom, that is the user can save, load, exit the game at any moment. The user can trade, mortgage/unmortgage properties as per their choice and status in the game.
- The rule button is also included in the game play so that new players can access the game rules at any moment during confusion.
- Most of our prototype incudes the player's property information, balance information on the screen to keep track of player's status in the game as the player can not remember all this information, so it is placed on the screen for recognition.

After all the discussion, we decided to remove save button from the main screen as there is nothing to save at that moment and this button is shifted to game play screen. We decided to show each property's information like the rent, houses, etc. whenever the player hover over it.

Additionally, in the group discussions, new design elements were introduced to further enhance the final UI. These elements included intuitive navigation to streamline gameplay, visually engaging graphics and animations to create an immersive experience, and comprehensive help and documentation resources accessible within the game for guidance and support.

By combining the strengths of each prototype, addressing their weaknesses, and introducing new design elements based on the group discussions, the final UI for the Monopoly game aimed to provide a user-friendly, visually appealing, and intuitive gaming experience, aligned with the UI design principles and the ten heuristics of the Nielsen Norman Group.

CMPT 276 Project Phase 2 Design Patterns

Factory

A factory design pattern was used/implemented to create the different types of tiles for the board. Since the different tiles are a group of closely related classes, and the creation of each tile object had to be done for all 40 tiles on the board, a factory was deemed as a good choice. The factory is used/implemented at the bottom of the board.py file, and is used in the Board class to create the board with a function called create_board(). This function takes in the file path of a json file that holds the data for each tile. The function simply loops through each json entry (data for a given tile), and passes the data as input to the create_tile() method in the TileFactory class (this class name was chosen for client code simplicity). The create_tile() method will then return the appropriate subclass of the Tile class (either a Street, Railroad, Utility, Tax, Chance, Community Chest, or Corner tile). The Tile class is an abstract base class that provides the interface for each subclass. The tile class includes an operation() method which is meant to be over-ridden by the various subclasses to define the specific concrete behaviour of that subclass. In conclusion, the creator class is the Tile class, the concrete creators are the various tile subclasses, and the products are the results of the operation() method in each subclass. All of this implementation and creation logic is encapsulated in the TileFactory class, to abstract the object creation process from the client code. The client code only needs to pass in a json file path, and the correct Tile object will be made for the board.

We chose this design pattern because there are common attributes for each tile type such as its name, position, and type, but the actual operation that needed to be performed for landing on each tile was different for each tile. With the factory, this can all be abstracted. The subclasses were necessary because each tile has unique attributes and functionality, therefore, they were necessary and were not just used to inherit the factory method. Furthermore, having the Tile class take care of the operation() for all cases would break dependency inversion, and having different functions for each type of tile would break inheritance. It is also easier to add new tile types by subclassing the Tile class, and adding data to the json file. This makes the code more flexible. The code is more reusable, since the creation logic is contained in the TileFactory class, and can be used elsewhere in the code. Lastly, since there were 40 tiles to create, the TileFactory class made the creation process easy.

This design improved the codebase in a number of ways. As mentioned before, the tile creation logic is more flexible, so modularity and extendibility were increased with this design. The TileFactory class allowed the overall codebase to be more readable. In the create_board() function, the code is simple, and the client does not need to know the details of creating the objects. Furthermore, the TileFactory class promotes the single responsibility principle, since its sole purpose is to create tiles.

State

We used a state design pattern in our code to identify the availability of certain buttons during the gameplay. Now we can enable or disable some buttons on the board. For example, the “roll dice” button should only be available when it is the player’s turn, and the player should not be able to click it during AI’s turn. It is a perfect opportunity for implementing the state design to the design, which introduces a way for an object to alter its behavior depending on the current situation. For now, we implemented state design to two buttons called “roll dice” and “end turn”. The player can only one of these two buttons at the same time.

We chose this design because we need a way to enforce the rule of the game. Using the previous example, if both “roll dice” and “end turn” are available for the player to click, then the player may keep moving by keep clicking “roll dice” or stay in one place by keep clicking “end turn”. This kind of rule-breaking action will break the basic rule of the game and may hugely affect the gaming experience. In the future, we plan to use this pattern in some other places as well. For example, the player can draw a chance or chest card by clicking the image of it (it is an “image button” in our code. Please reference our final gameplay UI above for its location). However, the function of the draw card is only available when the player’s piece locates at chance or chest block on the board.

This design improved the codebase by reducing the duplication of the code. By using this pattern, we do not need to make two classes for two classes for the same object but in different situations, such as “available button” and “unavailable button” that have the same design and only have the difference of whether it will handle an event.

Iterator

The codebase incorporates the iterator behavioral design pattern in the 'tiles.py' section to handle the iteration over card lists, specifically in the chance and community chest components. There are 2 class interfaces that are necessary to implement the iterator design pattern iterator and iteratorCollection. These are represented in the code as Iterator and Card_Collection. From these classes there are a collection class and an iterator class for each chance and community chest. There are different classes for chance and community collections and iterators since they have varying get out of jail cards values across the 2 decks. Most of the code is similar to a regular implementation of an iterator with a list but there are 3 things that are different from a regular iterator that are implemented. The first being the get_next() method of the iterators calling fill_list() when the collection is empty. fill_list() will take 16 numbers that represent cards and try to add them to the collection. The second being it won't add a jail_card item if the jail_card is in play. The collections know the card is in play since when calling get_next() it will set the variable as true. Finally, as the chance and community chest decks are designed to have an unlimited number of cards, the 'has_next()' method of the iterator always returns true.

The focus of the chance and community chest tiles is the next card method and the iterator provides the ideal solution as it has the 'get_next()' method to retrieve the next card. It will abstract how the get_next() is implemented making it easier to call it without worry of running out.Without the iterator pattern, an alternative approach would involve explicitly managing a list of cards and manually refilling it each time it becomes empty. Additionally, the contents of the list would need to be adjusted based on whether the "get out of jail" card is in play or not. It being a single class means we could implement the singleton design pattern so there would not be any confusion between the tiles on which card is being drawn next.

By utilizing the iterator pattern, a lot of complexities are abstracted away, allowing the implementation to encapsulate the refilling and card selection logic internally. This simplifies the codebase and provides a cleaner interface, where the only concern is when to get the next card using the 'get_next()' method.