Monopoly

- Include at least 6 themes
- Include at least 3 rulesets

Features

- Roll the dice
- Click on different positions on the board
- Pick a card from the chance/community chest decks
- Buy properties, or add houses/hotels to properties
- Mortgage properties
- Bid on unpurchased properties

Individual Responsibilities

- Cameron: Spaces (ownable, go, etc), Players, Rulesets, Cards, AI strategies
- Lina: Board, secondary: frontend - finding images for each space, CSS, Buying properties, New rulesets
- Delaney: Cards, Players, Ruleset, Paying Rent, Cards, Design Coach issues
- Grace: Frontend/GamePlay/GameView, Taking a turn, CSS
- Anna: Rules interface, DataReader, General frontend support, Languages, House building, AI strategies
- Everybody: One theme

Sprint Breakdown

- First sprint: One Monopoly board with the basic rules, a basic layout, and as many as three players included. Players can choose their piece, roll the dice, access their own money, track their properties, houses, hotels, move around the board, see properties, and take community chest/chance cards
- Second sprint: Each add a new visual theme, A new ruleset, New design features (buttons to change the theme), Up to 4 players
- Third sprint: Any combination of rules and themes will work, Multiple players can play, AI players can be added

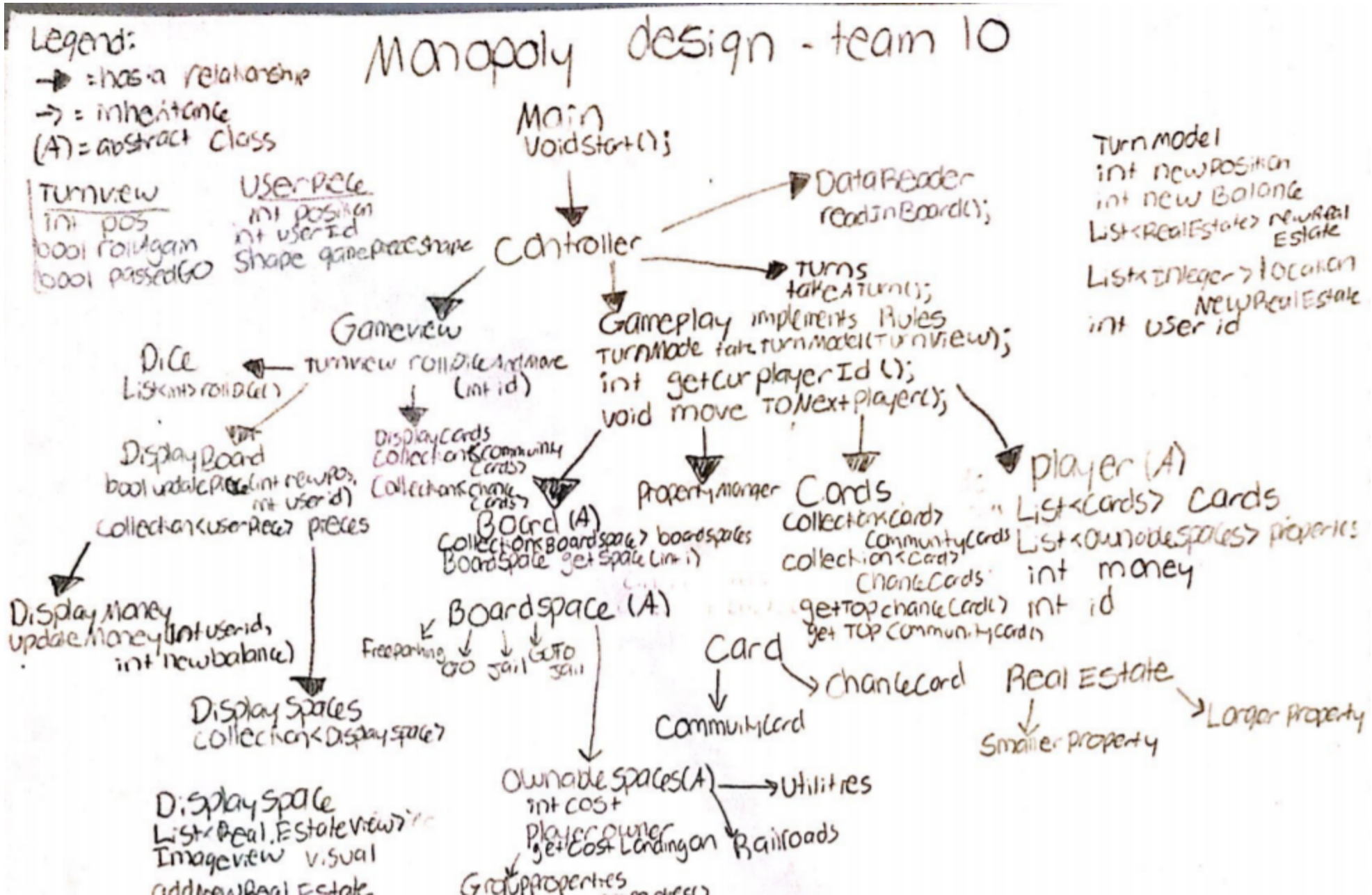
User Interface Wireframe Demo

Working Code Demo

Design and Framework Goals

- Different board setups, different cards, different types of houses, different genres, different languages, different stylesheets, rules such as betting when you land on something

- Dependencies
- Encapsulation



GameView

- Purpose: Calls all the necessary classes and methods to create the display of the board. For example, calls the ButtonsMaintainer and DisplayBoard
- Extension: Referencing and images
- User Support: Allows them to follow single responsibility and open closed principle

Cards

- Purpose: Reshuffling the cards when the deck needs to be reshuffled and letting the user always take the top card off the deck
- Extension: Different types of cards easy to add via referencing and new class --> This provides for extension because different cards can easily be added to the Decks.
- User Support: Allows them to follow single responsibility and open closed principle

First Use Case

Player X places a house on their property on their turn

```
\\ In the TurnModel class:
        if(playerOne.canPlaceHouse()) playerOne.placeHouse();

\\In the Player class, canPlaceHouse() method:
        \\(color neighborhood we are checking is blue)
        int numBlueProperties = 0;
        for(OwnableSpace property: propertyList){
                if(property.neighborhood().equals(BLUE)) numBlueProperties++;
        }
        if(numBlueProperties == FULL_NEIGHBORHOOD) return true;
```

Second Use Case

Player X has no money and has mortgaged all properties

```
// in the player class
        boolean determinePlayerEliminated() {
                return balance == 0 && checkAllPropertiesMortages();
        }

// in the turns class
        while (!players.isEmpty()) {
        // do the turn loops and at the end of the turn:
                if (currentPlayer.determinePlayerElimated()) {
                players.remove(currentPlayer);
                }
        }
```

Alternative Design Choice

Design 1: Keep the content of the card in model and view:

- Pros: Easy access in both classes
- Cons: Waste of storage and model might not know how to interpret the information- view knows the language

Design 2: Keep the content of the card in model and view, but in model only give Card class the information it needs like spot to move to or cost

- Pros: model only has what it needs which is good
- Cons: harder to implement and wastes space

Design 3: Keep the content of the card in view and give it to model

- Pros: doesn't waste space
- Cons: how will model know what to get from the card