Design goals

- Many values are written in the properties file
- All major classes are abstract and their subclasses can be specified in properties
- Default starting properties for when player hasn’t uploaded a game yet
- Defaults to constants if you don’t enter a value or enter it incorrectly

API 1: ClassicGamePlay (originally GamePlay)

```
public class ClassicGamePlay extends GamePlay implements TurnableModel {

        public void initializeGame(Map<Integer,String> playerIdsAndTypes, InfoForGamePlay infoForGamePlay, TurnableView turnableView, Promptable promptableIn, ErrorPrintable errorPrintable);
        public PropertyUpgradeable getPropertyUpgradeable();
        public Player getCurrentPlayer();
        public void moveToNextPlayer();
        public int getCurrentPlayerId();
        public void buyRealEstate(DisplayInitializer displayInitializer, InfoForGameplay infoForGameplay);
        public void makeTrade(DisplayInitializer displayInitializer, InfoForGameplay infoForGameplay);
        public boolean determineGameOver();
        public int getWinningPlayerID();
        public void makePlayerUsingDiceRoll(List<Integer> diceRoll);
        public void movePlayerToSpace(int locationUpdate);
        public int getNumPlayers();
        public Optional<TurnModel> doActionOnCurrentLocation(int playerID, int spaceID);
        public void startPlayingGame(int playerId);
        public void startGame(TurnableView turnableView, Optional<String> startPlayerType, Promptable promptable);
        public boolean determineStartingPlayerId(List<Integer> diceRoll);
        public Board getBoard();
        public ModelDiceRollable();
        public Promptable getPromptable();
        public Map<Integer,Player> getPlayers();
        public Cards getCards();
}
```

API 1: ClassicGamePlay (originally GamePlay)

- Flexibility
- Facilitating good code
- Change throughout sprints

API 1: ClassicGamePlay Use Cases

- Controller

```
    gamePlay.initializeGame(playersIdsAndTypes, dataReader.getInformationForGamePlay(),
        gameView, gameView.getPromptable(), gameView.getErrorPrintable());
    gameView.initializeGame(gamePlay.getModelDiceRollable(),gamePlay.getPropertyUpgradeable(),
        dataReader.getInformationForGamePlay());
```

- ComputerPlayer

```
    protected void turnToRoll(TurnableView turnableView, ModelDiceRollable modelDiceRollable,
      TurnableModel turnableModel) {
        List<Integer> diceRoll = modelDiceRollable.rollDice();
        turnableModel.movePlayerUsingDiceRoll(diceRoll);
        this.promptable.alertOfComputerPlayerRoll(diceRoll,super.getId());
}
```

API 2: Cards

```
public abstract class Cards {

 public abstract Card getTopChanceCard();
 public abstract Card getTopCommunityCard();
 public abstract void shuffleChanceDeck();
 public abstract void shuffleCommunityDeck();

}
```

API 2: Cards

- Flexibility
- Facilitating good code
- Change throughout sprints

API 2: Cards Use Cases

- ClassicGamePlay

```
        cardDecks = new CardsFactory()
    .createCards(infoForGamePlay.getCardsType(), infoForGamePlay,
        errorPrintable);
```

- NonOwnableChanceCardSpace

```
        Card cardPicked = this.getCardDecks().getTopChanceCard();
player.drawCard(cardPicked.getId());
super.performCardAction(turnModel, player, cardPicked, spaceID);
```

Design that has remained stable (Board display)

- Board graphics
- We originally planned on reading in the board graphics as individual files from data

```
@Override
public List<String[]> getBoardSpaceImagePaths(File csvFile) throws FileNotFoundException {
 List<String[]> boardSpaceImages = new ArrayList<>();
 List<String[]> boardData = readFile(getStream(csvFile));
 String dataPath = boardData.get(0)[0];
 for (int i = 1; i < boardData.size(); i++) {
   String[] imageFile = new String[2];
   imageFile[0] = dataPath + boardData.get(i)[0];
   imageFile[1] = boardData.get(i)[1];
   boardSpaceImages.add(imageFile);
 }
 return boardSpaceImages;
}
```

- We made classes in view to store and display the graphics

Design that changed a lot (Players)

- Evolved from handling player statistics to helping trigger communication based on the player
  - Triggering next step for the player solved the issue of turn classes having too many responsibilities
  - Next actions by players are immediately triggered instead of waiting on method call handling
- Extracted money transactions from Player to Bank
  - Takes a responsibility away from player
  - Significantly cleaner design-wise

Contrast with initial wireframe and sprint plans

Initial wireframe vs. complete

- The project grew in scale and flexibility immensely, so the contrast of the initial wireframe to the complete version reflects this
- More front-end features than originally documented on the wireframe

Sprint plans vs. Reality

- Most components finished on schedule
- Interactions were handled mostly in the end as we reevaluated how they would work based on the growing size of the project and refactoring

What we each learned from Agile/Scrum process

- Cameron: I underestimated the usefulness of Git issues. As we continued to use them more effectively throughout the project, it became so much easier to communicate because we knew who to talk to about specific parts of the project we needed more information on.
- Anna: The importance of daily standup
- Delaney: It is really important to predict time needed to work through bugs in creating realistic sprint deadlines
- Grace: I learned that it is important to be realistic rather than hopeful on the sprints to better manage your time
- Lina: I learned that planning ahead, especially on front-end graphics, is really important to having an organized timeline. When we worked on our graphics, it was significantly easier to implement the views we had planned out well (like the board and the players) than the views we had not planned for (like the pop-ups).

Four significant events

1. Finishing the classic monopoly board graphics
2. Making the decision to use Banks to store money in our game
3. Writing the Promptable interface
4. Abstracting all of the highest levels of classes

What we each learned about managing a large project

- Cameron: It’s important to be realistic about how much can be accomplished in each set timeframe (sprints in this case)
- Anna: The importance of thinking ahead and planning out classes for every conceivable purpose
- Delaney: For a big project with many interactions, plan to communicate and work collaboratively more than problem solving alone
- Grace: I learned that it is very important to communicate what you are about to work on because that allows for no duplicated work and you can ask group members about their code later on.
- Lina: I learned that it’s important to have oversight about how the different parts of the project work together, and to communicate with the people that are working on interconnecting code.

What we worked to improve on

- Fundamentals of software design
- Day to day communication
- Timeliness of sprint goals

What could still be improved

- Meeting scheduling and regularity
- Mutual encouragement
- Breaking out of design schemas

One thing we each learned about creating a positive team culture

- Cameron: Meeting over Zoom is much more valuable than just texting because you can develop a rapport that can be much more useful
- Anna: The importance of reliability and responsibility in building team trust
- Delaney: This encourages frequent communication on the status on our work in the project regardless on whether they are positive or negative updates, which is really important to maintain
- Grace: I learned that it is important to help others with their parts of the code if they have a bug even if you are not that familiar with the code
- Lina: I learned that reaching out to people with questions about their code is really important on a project on this scale.

Still useful from Team Contract

- Communicating via text
- Updating the team when merging
- Working via consensus

To update in Team Contract

- Sunday deadlines
- Increasing frequency of video meetings
- Team meeting and behavior contract

What we each learned about communicating and solving problems collectively

- Cameron: Meeting with just one or two people working on similar aspects to those you're working on can be really helpful in making sure all of the parts of the project work well together
- Anna: Reliability and responsibility are incredibly important for collective problem solving
- Delaney: Constant communication is integral over a long term project to ensure that we are meeting goals and deadlines. Problem solving collectively, especially in a big group, helps refine ideas to the best possible design and integration
- Grace: I learned it is important to explain a problem very well so that the whole team is able to work on it
- Lina: It is really valuable to problem solve together on large-scale projects.

Conclusions

We enjoyed this project!

- Code development: Learning to develop on a larger scale
- Teamwork: Working remotely with a larger team
- Creativity: Developing themes based on each of our interests