Progress Report

Simplified Monopoly

Specification

Our goal is to create a simplified version of the popular real-estate board game, Monopoly. Players participate in a game with up to three other players. Before the game begins, players can set a net worth goal and a maximum number of rounds. The objective of the game is to be the first player to reach this net worth goal within the specified maximum number of rounds. If all but one player has gone bankrupt, the remaining non-bankrupt player wins. Each player starts with $1000 in cash and players take turns rolling a die, moving across the board, interacting with special board tiles (Jail, Auction, Surprise tiles), paying rent to other players, and buying, selling, auctioning, trading, and upgrading properties (cities and public properties) until the game ends.

CRC Model

User Interface: CMDLineUI // takes in input from the user from options of moves they can make

Controller: Game Controller // Controls game state gives orders to use cases

Use cases: Property Manager, Money Manager, Board Manager, Auction Manager // These take orders from the game controller and accesses elements in entities to update what needs to be done, maybe give some examples

Entities: Board, Tile, PropertyTile, SpecialTile, City, PublicProperty, Building, Player, Token // Objects representing a set of critical business rules/data

Scenario Walkthrough

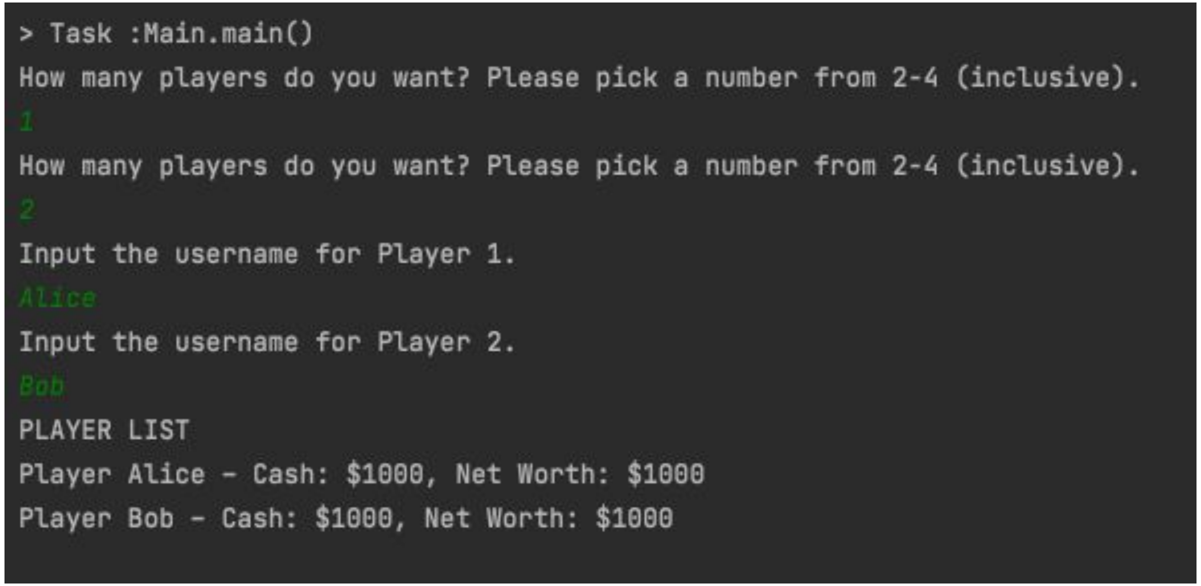
When the program is run, the command line (**CmdLineUI**) prompts the user to input the numbers of players in the game (between 2 to 4 inclusive). Next, it asks the user to enter the usernames of each player. Then the **GameController** starts and runs the game. At the beginning of a game, the **GameController** works with the **BoardManager** to initialize the players (**Player** objects) and put the players in the starting locations. Once this is set up, a string containing the name, cash, and netWorth of each player (the respective attributes of the **Player** objects) is displayed.

Skeleton Program Overview

For this program, we implemented CmdLineUI (framework), GameController (our controller), BoardManager (a use case), Player and Token (both entities).

In Main.java, we create an instance of our GameController and our CmdLineUI. We call CmdLineUI’s runPlayerSetup() to prompt the user for the total number of players and the usernames of each player and to call the controller’s runPlayerSetup() method. The GameController stores an instance of BoardManager, whose addPlayers() method is called to construct a new Player with each username. Then, we print a string representation of the controller’s BoardManager from the runPlayerSetup() method, which contains a list of the players as described in the walkthrough.

Skeleton Program Output



```
> Task :Main.main()
How many players do you want? Please pick a number from 2-4 (inclusive).
1
How many players do you want? Please pick a number from 2-4 (inclusive).
2
Input the username for Player 1.
Alice
Input the username for Player 2.
Bob
PLAYER LIST
Player Alice - Cash: $1000, Net Worth: $1000
Player Bob - Cash: $1000, Net Worth: $1000
```

What has each group member been working on and what do we plan on doing next?

- Jacqueline – Worked on specification and skeleton program. Going forward, will continue to refine CRC model and gradually implement remaining classes/interfaces.
- Danny- Worked on specifications and CRC cards. Will continue to implement classes & interfaces in order to ensure smooth interactions between classes.
- Parth - Worked on the scenario walkthrough and CRC cards. Will continue to implement classes and interfaces, will work on finalizing CRC cards, if needed, so that they completely follow clean architecture

What has worked well so far with our design?

1. We have different use cases handling different needs of the program (such as PropertyManager, BoardManager, AuctionManager, and MoneyManager) and they all have low coupling and high cohesion which is desired.
2. Clear separation of classes into entities, use cases, controllers, and UI according to the dependency rule and the principles of clean architecture.
3. Use of inheritance for tiles, properties, cities, and public properties aligns well with our intuitive understanding of the relationships between these objects in a standard Monopoly game.

Open questions we’re struggling with...

1. We are having some issues with IntelliJ and test cases clashing with gradle for which we had to find a fix in settings through stack overflow, is this supposed to happen or did we mess something up.
2. Whether use cases can interact with other use cases, or should the controller interact with them.
3. How should we set up Input/Output boundaries for the controller’s interactions with each manager?