Requirements Documentation

Introduction:

The game we are implementing is Monopoly, a turn-based game where players roll a dice to move around on a finite board and use money to purchase properties. Players who land on other people's property must pay "rent", and a winner is determined when all players have gone bankrupt except for one.

User Stories:

- As a player, I want to be able to select how many friends I am playing with, and have the option to play with AI opponents if there are not enough people present.
- As a player, I want the dice rolls and movement to be randomized so the game feels fair and unpredictable.
- As a player, I want to encounter random chance cards during the game which may benefit or harm me, as it adds an element of surprise and unpredictability to the game.
- As a new player, I want a tutorial to be present, either in a written or interactive format, to understand the rules and gameplay mechanics of Monopoly.
- As a player, I want to be able to save the game at any point, so I am able to resume my session when I return.

Functional Requirements:

- System asks how many players would like to play + the number of AI players
- Have a rational AI which does not immediately purchase property just because they are able to
- Interactive GUI to play the game on
- Save system (auto saving either on events or periodically?)
- Randomization for dice rolls and chance/community chest cards.
- Tutorial must be present either written or interactive.
- Set of finite chance/community chest cards which may benefit or hurt whoever has drawn them.

Non-functional Requirements:

- AI determination must be fast (instantaneous)
- Java (OOP)
- Different classes for game objects (player, AI, property, event space, card, etc.)
- Double click executable.
- Code AI as an expert system?
- “Board” class used to construct game according to certain parameters.
- GUI built using JavaFx/Swing
- Metadata file for saving the state of the game based on minimum number of parameters.
- Check if the game is over at end of each turn.

UML Diagram: