Project Proposal: Java Monopoly

Description

The program seeks to implement a playable version of the board game, Monopoly.

Intended user

Someone who wishes to play Monopoly. (?)

Problems to solve

Data

A game of Monopoly can be decomposed into a large data structure, and this comprises the core of this project. A set of objects which contain certain attributes are affected throughout gameplay, either randomly, or by user decision.

The primary element of the game data is the board. It contains sets of information about both the players and board spaces. Both the players and spaces are represented in an `ArrayList`. In the case of spaces, this `ArrayList` can contain several different object types, all derived from the base `Space` class. Consequently, each space can have its own methods that are called during a generic event, e.g., calculating rent.

CPU players

Implementing an extremely basic CPU player is trivial, wherein the player automatically performs its necessary actions, and executes any required decision as `true`.

A more advanced CPU player can be manifested through several styles:

Random probability: For each mandatory binary decision that the player encounters, a probability is assigned to one option. When an action is required, the execution of this decision is dependent on its probability. This effectively makes all actions random, where the probability could be determined at the start of the game by a seed.

Basic statistical model: When running simulations of Monopoly, particular patterns begin to appear. For example, the Jail space is often a ‘trap’ for players - this means that the spaces immediately following the Jail space have the highest probability of landing out of any other spaces on the board.

As a consequence of this land-probability distribution, some properties have a measurably higher return on investment (ROI) when improvements are constructed on them.

A statistically-based CPU player could make decisions similarly to that of the random probability model, but with a multiplicative factor based on the estimated ROI of a particular space.

Machine-learning model: While an ML-based model would undoubtedly perform very well compared to the other models listed, this is simply out of scope for this project. An implementation could be possible by using a relatively simple training method such as (framework here)

An important note to make is that much of a player’s success in Monopoly is simply luck. The dice roll is the primary engine of the game, and there is an inherent amount of randomness that cannot be surpassed entirely with skill. One benefit to this is that the ability of any CPU player is not entirely dependent on its code - the luck will sometimes skew the game state in its favor regardless.

Technologies needed

The primary data of the application is stored in a single serializable class, `Board`. This class can then be saved to a file to preserve the game state, and can be used by a future instance of the program to resume gameplay. Basic file I/O is needed as a result of this functionality.

A GUI front-end is provided to the user. This indicates much of the essential information needed for gameplay, including:

- All properties and game event spaces that exist on the board

- Position of each player
- Balance of each player
- Whose turn it is
- Whether a specific property is owned, and if so, by whom
- A log containing information about all previous turns, and the actions taken during those turns

The GUI additionally provides mechanisms for controlling the central data structure of the game. Examples of actions through this interface include:

- Rolling dice
- Ending turn
- Viewing information about a specific property