Monopoly Game

Overview

This project provides a streamlined Monopoly game experience. Players take turns moving around a board based on dice rolls, buying unowned properties, paying rent to property owners, and aiming to avoid bankruptcy. The game automatically processes all turns and determines the winner based on remaining funds when the game ends.

Features

- Dynamic Board Loading: The game board is loaded from a board.json file, which allows flexibility in board design.
- Configurable Dice Rolls: Dice rolls are preloaded from JSON files (rolls_1.json and rolls_2.json), with user selection at runtime.
- Game Flow: Players take turns to move. Passing the starting position ("GO") rewards players with money.
- Property Management: Players can purchase properties, pay rent, and even achieve monopolies that double rent, and players have property attributes.
- Bankruptcy Handling: Players with no money are eliminated from the game, and the game ends if a player goes bankrupt.

Setup Instructions

Environment recommandation:
- Mac Book Air M1
- IOS 14.5 for Mac
- Ruby 3.4.1
- Visual studio code with Ruby LSP plugin installed

Clone the Repository or directly open project file:
- https://github.com/JunyIIIChen/Woven-Monopoly.git

Prepare Required Files:

- Ensure board.json and dice roll files (rolls_1.json and rolls_2.json) are present in the project directory.

How to Run

Start the Game: Run the game script using the following command:
- cd Woven_Monopoly
- ruby Monopoly.rb

When prompted, select dice rolls file (1 or 2):
- input 1 or 2

View Results: results, including the winner and player standings.

Testing

Running Tests
Unit tests are included to verify key functionalities.
To run the tests, execute:
- cd Woven_Monopoly
- ruby test/test_monopoly.rb
- ruby test/test_property.rb
- ruby test/test_player.rb
- ruby test/test_board.rb

Test Coverage:
- Unit test and integration test, game initialization, including loading of board and dice rolls.
- Player actions like moving, purchasing properties, and paying rent.
- Rent calculation, including scenarios with monopolies.
- Bankruptcy handling and winner determination.

Design Decisions

Object-Oriented Structure:
- Core components such as Player, Property, Board, and Monopoly are encapsulated in dedicated classes to ensure modularity, reusability, and easy debugging.

JSON Configuration:
- The game board and dice rolls are stored in JSON files, enabling non-technical users to modify game configurations without changing the source code.

Gameplay Flow:
- The turn-based system ensures fairness by iterating through players in a round-robin manner.
- Board design: Using array type data ensures the board wraps around.

Early Game Termination:
- If any player goes bankrupt, the game ends