PROGRESS REPORT OF 2048 GAME

The goal of this project is to develop an AI-powered version of the 2048 game using **Monte Carlo Tree Search (MCTS)**. The game allows two AI agents to compete against each other making strategic moves to maximize their scores. The AI evaluates possible moves through simulations and selects the best action based on expected rewards.

The below project files are updated to achieve the different aspects of the game:

1. **problems.py** – Contains the **2048 game mechanics** (grid management, movement, merging, game-over conditions).
2. **algorithms.py** – Contains the **MCTS-based AI agent** to play the game.
3. **demo.py** – Runs a **two-player AI match** where agent play alternately until the game ends.

1) problems.py – 2048 Game Mechanics:

This file defines the core game logic, including:

- **Game Grid:** A **4×4 board** where tiles are moved and merged.
- **Tile Movement:** Implements left, right, up, and down moves.
- **Tile Merging:** When two adjacent tiles have the same value, they merge into one.
- **Random Tile Addition:** After every move, a new tile (2 or 4) is added at a random empty position.
- **Game Over Check:** The game ends when player reaches **2048** (win condition), No valid moves are left (lose condition).

2) Algorithms.py – Monte Carlo Tree Search (MCTS) AI:

This file implements an AI agent that plays the game using the **Monte Carlo Tree Search (MCTS)** approach.

MCTS Works as below

1. **Simulate Random Moves:** The agent picks a move (left, right, up, or down) and simulates multiple random moves.
2. **Evaluate Board Score:** The AI calculates the total score (sum of all tile values) to determine move effectiveness.

3. **Select the Best Move:** The move leading to the highest expected score is chosen.

3) demo.py – Running AI vs. AI Matches:

This file runs a 2048 game where two AI agents compete. It alternates turns until:

- A player reaches 2048 (win condition).
- No more moves are possible (game over).

Game Mechanics – Implemented tile movements, merging, random tile placement, and game-overchecks.
AI Agent Developed – The MCTS-based AI selects optimal moves through simulations.
Two-Player Mode Implemented – Two AI agents take turns until one reach 2048 or the gameends.
Successful Testing – The AI successfully plays multiple games, showing smart decision-making.

Performance Optimization:

a. Simulation Efficiency

Currently, the **MCTS agent** simulates moves by randomly selecting them and applying them to a game copy. To make the AI more efficient, several optimizations can be implemented as listed below

- **Early Termination:** Stop the simulation if a certain **threshold score** is reached (e.g., 2048 or higher).
- **Move Pruning:** Prioritize moves that are more likely to lead to higher scores based on the current game state (e.g., avoid moves that lead to tile congestion).
- **Parallelization:** Run multiple simulations concurrently to speed up the decision-making process.

b. Monte Carlo Tree Search Enhancements

MCTS can be enhanced by:

- **Using UCB1 (Upper Confidence Bound for Trees):** This can help balance **exploration** and **exploitation** during simulations, leading to better move selection.
- **Improved Selection Strategy:** Instead of selecting random moves, the agent could be more strategic in choosing branches of the game tree that have historically performed well.

GITHUB LINK:

https://github.com/INFO-450-550-Artificial-Intelligence/final-project-swethakusampudi25.git