YONMOQUE-HEX

ARTIFICIAL INTELLIGENCE PROJECT PRESENTATION

Miguel Tomás Vieira Rodrigues | 202205749

OVERVIEW

- PROBLEM
- ALGORITHMS
- EXPERIMENTAL RESULTS
- CONCLUSIONS
- REFERENCES

PROBLEM

In this project, we are supposed to develop Yonmoque-Hex, a two-player **Adversarial Game:**

- Board Game played on a **hexagonal grid**.
- Players take turns placing or moving pieces to form a **4-in-row** to win
- Avoid **5-in-row**, which results in a loss
- Key mechanics are: **Placement** and **Movement**

The main objective of the project is to:

1. Develop a playable Yonmoque-Hex implementation with **Person vs Person**, **Person vs Computer** and **Computer vs Computer** modes; GUI for interactive gameplay and Adversarial AI using Minimax and Monte Carlo Tree Search
2. Compare Algorithm Performance

PEAS ANALYSIS

PERFORMANCE MEASURE

Primary: Win by forming a **4-in-row** of the player's color
Secondary:
- Avoid creating a **5-in-row**
- Minimize opponent's alignment opportunities
- Maximize efficiency of moves

ENVIRONMENT

Fully Observable: All pieces and board states are visible
Deterministic: Rules are fixed
Sequential: Player alternates turns
Dynamic: Board state changes with each move
Discrete: Finite board positions and actions
Multi-Agent: Two adversarial players

ACTUATORS

Place: Place a piece from reserves (pieces out of the board) onto an empty tile
Move: Slide a piece along valid directions (vertical, horizontal, diagonal)

SENSORS

Board State: Positions and colors of all pieces on the grid
Reserve Status: Number of remaining pieces for each player
Valid Moves: Legal placements and movements based on tile colors and occupied tiles

ALGORITHMS

MINIMAX

- Adversarial search algorithm for two-player games
- Assumes the opponent plays optimally and alternates between maximizing and minimizing the player's advantage
- Key features:
  - Depth-limited Search: Explores possible moves up to a fixed depth
  - α-β cuts: Optimizes by eliminating branches that cannot influence the final decision
  - Heuristic Evaluation: Uses evaluate_board() to score non-terminal states
- Works better for: Deterministic games with low branching factors; Scenarios where a strong heuristic can guide decisions; Smaller game Trees

MONTE CARLO

- Simulation-based algorithm that estimates move quality through random playouts
- Key Features:
  - Four phases: Selection, Expansion, Simulation, Backpropagation
  - Difficulty Levels: Easy, Intermediate, Hard
  - Adaptative: Excels in games with high-branching factors
- Works better for: Games with High Branching Factors; Problems with no clear heuristic or complex state evaluations; Unpredictable environments where random simulations improve adaptability

RESULT

- Both algorithms tested across 4 game-scenarios with identical parameters.
- The minimax achieved a better win rate
- For real time play, it's better to use MCTS-Intermediate as it balances speed and skill
- For analysis, it's better to use Minimax-Hard for optimal moves
- We shall avoid using Minimax-Hard in time-constrained scenarios

MINIMAX

Difficulty	Avg.Move Time	Avg.Moves per Game
Easy	0.05-0.07	4-13
Intermediate	0.57-0.93	4-8
Hard	4.87-11.59	4-9

MONTECARLO

Difficulty	Avg.Move Time	Avg.Moves per Game
Easy	0.63-0.99	6-15
Intermediate	1.63-1.66	4-7
Hard	5.99-9.41	5-12

CONCLUSION

This project highlights the importance of aligning algorithm choice with game complexity and real-time requirements.
Minimax and MCTS represent two pillars of adversarial reasoning – one rooted in classical optimality and the other in modern adaptability. By mastering their strengths and limitations, we pave the way for AI systems that are not only intelligent but also intuitive, responsive and deeply aligned with human gameplay experiences.

REFERENCES

- https://www.freecodecamp.org/news/minimax-algorithm-guide-how-to-create-an-unbeatable-ai/
- https://builtin.com/machine-learning/monte-carlo-tree-search

THANK YOU

ARTIFICIAL INTELLIGENCE PROJECT PRESENTATION

Miguel Tomás Vieira Rodrigues | 202205749