Chapter IV - Adversarial Search

4.1 Games

Games are a common and useful domain for testing artificial intelligence techniques. In AI, a game is often modeled as a search problem where agents (players) take turns making decisions to maximize their chances of winning.

Types of Games:

- **Deterministic Games**: Games with no element of chance (e.g., Chess, Tic-Tac-Toe).
- **Stochastic Games**: Games that involve randomness (e.g., Poker, Backgammon).
- **Perfect Information Games**: Games where all players have full knowledge of the game state (e.g., Chess, Checkers).
- **Imperfect Information Games**: Games where players have incomplete knowledge (e.g., Poker).

Game Representation:

- **Initial State**: Describes the starting configuration.
- **Players**: Defines who plays the game (e.g., MAX and MIN in two-player games).
- **Actions**: The possible moves each player can make.
- **Transition Model**: Defines how moves change the game state.
- **Terminal State**: Defines when the game ends and assigns a value (win, lose, draw).
- **Utility Function**: A numerical representation of game outcomes (e.g., +1 for win, 0 for draw, -1 for loss).

4.2 Optimal Decisions in Games

To make optimal decisions in adversarial environments, AI agents must evaluate possible future moves and choose the best one. The **Minimax Algorithm** is a fundamental method used in perfect information games.

Minimax Algorithm:

- **MAX Player**: Tries to maximize the score.
- **MIN Player**: Tries to minimize the score (acting as an opponent).

- **Tree Representation**: The game is represented as a tree where nodes represent possible game states, and edges represent moves.
- **Backtracking**: The algorithm explores all possible outcomes and propagates values back to the root to determine the best move.

Example:

- In a Tic-Tac-Toe game, the minimax algorithm simulates every possible move and selects the one leading to the best outcome.

Limitations:

- **Computationally Expensive**: In games with large branching factors (like Chess or Go), minimax can become impractical.
- **Cannot Handle Uncertainty**: It assumes a fully deterministic environment.

4.3 Alpha-Beta Pruning

To improve minimax efficiency, **Alpha-Beta Pruning** eliminates branches in the game tree that cannot possibly influence the final decision.

Key Concepts:

- **Alpha (α\alpha)**: The best value MAX can achieve so far.
- **Beta (β\beta)**: The best value MIN can achieve so far.
- **Pruning**: If a node's evaluation proves that it won't be selected, further exploration is unnecessary, reducing computations.

Effectiveness:

- Reduces the number of nodes evaluated, improving efficiency.
- In an ideal case, it reduces the complexity from O(bd)O(b^d) to O(bd/2)O(b^{d/2}), where bb is the branching factor and dd is the depth.

Example:

- In Chess, if one move is found to be significantly worse than another, we stop evaluating that branch early.

4.4 Imperfect Decisions in Real-Time

In real-world scenarios, AI does not have unlimited time to search for the perfect move. Instead, it must make **imperfect but reasonable decisions** under time constraints.

Techniques for Real-Time Decision Making:

1. **Depth-Limited Search**: Search is stopped at a specific depth rather than reaching terminal states.
2. **Heuristic Evaluation Functions**: Instead of computing exact outcomes, heuristics estimate the value of a game state.
3. **Iterative Deepening**: A combination of depth-first and breadth-first search that allows deeper analysis as time permits.
4. **Monte Carlo Tree Search (MCTS)**: Simulates multiple possible game playouts to guide decision-making (widely used in Go and modern game AI).

Example:

- In real-time strategy games like StarCraft, AI must make quick decisions based on incomplete information.

4.5 Stochastic Games

Stochastic (or probabilistic) games introduce elements of chance, requiring AI to handle randomness in decision-making.

Examples:

- **Dice-based Games**: Backgammon, Monopoly.
- **Card Games**: Poker, Blackjack.

Solution Approaches:

1. **Expectimax Algorithm**: A variation of Minimax that considers probabilistic outcomes. Instead of choosing the maximum or minimum value, it computes the expected value.
2. **Monte Carlo Methods**: AI simulates thousands of random game scenarios to estimate the best move.
3. **Reinforcement Learning**: AI learns optimal strategies by playing many games and adjusting based on outcomes.

4.6 Partially Observable Games

In **partially observable games**, players do not have full knowledge of the game state. They must infer missing information based on observations and probabilities.

Examples:

- **Poker**: Players do not know opponents' hands.
- **Battleship**: Players only receive limited feedback about opponent actions.

Approaches to Handling Partial Observability:

1. **Belief State Representation**: Instead of a single state, AI maintains a probability distribution over multiple possible states.
2. **Hidden Markov Models (HMMs)**: Used for tracking hidden information in dynamic systems.
3. **Bayesian Networks**: Helps AI make decisions under uncertainty by incorporating probabilities.

4.7 State-of-the-Art Game Programs

AI has achieved remarkable success in various competitive games:

1. Chess (Deep Blue, 1997)

- IBM’s **Deep Blue** defeated world champion Garry Kasparov using brute-force search and evaluation functions.

2. Go (AlphaGo, 2016)

- **AlphaGo** defeated human grandmasters using deep neural networks and reinforcement learning.

3. Poker (Libratus, 2017)

- **Libratus** outperformed professional poker players using game-theoretic reasoning and self-play.

4. StarCraft II (AlphaStar, 2019)

- **AlphaStar** reached **Grandmaster level** using deep reinforcement learning.

5. Dota 2 (OpenAI Five, 2018)

- **OpenAI Five** competed against professional players using reinforcement learning and self-play.

4.8 Alternative Approaches

New AI techniques continue to evolve beyond classical search and reinforcement learning.

1. **Deep Learning-Based Agents**
   - Use **convolutional neural networks (CNNs)** and **transformers** to process game data and predict actions.
   - Example: **AlphaZero** trained entirely through self-play without human input.
2. **Neuroevolution**
   - Uses **genetic algorithms** to evolve AI agents.
   - Example: **NEAT (NeuroEvolution of Augmenting Topologies)** optimizes neural network structures for better decision-making.
3. **Hybrid Approaches**
   - Combines **rule-based systems** with **machine learning** for flexible strategies.
   - Example: AI for **real-time strategy games** often blends symbolic reasoning with deep learning.

Exercises

1. **Minimax Algorithm:** Implement a minimax-based Tic-Tac-Toe AI. How does its performance change with increasing depth?
2. **Alpha-Beta Pruning:** Compare the execution time of a minimax algorithm with and without alpha-beta pruning.
3. **Expectimax:** Implement an Expectimax AI for a simple dice-based game (e.g., Pig). How does it handle randomness?
4. **Monte Carlo Tree Search (MCTS):** Implement a basic MCTS algorithm for Connect Four. How does the number of simulations affect performance?
5. **Partially Observable Games:** Design an AI for a simple card game where it must infer hidden cards from observed moves.