AI Assignment II

# Kabaddi Game

Ashish Mishra
MT24020

## 1. Overview

This report presents a analysis of artificial intelligence agents designed to compete in a custom, grid-based Kabaddi simulation. Our solution implements and compares four distinct AI agents: a **Random** agent (for baseline performance), a **Greedy** agent (for heuristic-driven decision-making), **Alpha-Beta Search** (for theoretical optimality), and **Monte Carlo Tree Search (MCTS)** (for practical, simulation-based performance). The program simulates the game in two separate environments: a deterministic **turn-by-turn** mode and an uncertain **simultaneous-move** mode. The final output is a detailed comparative analysis of each agent's performance and strategic capabilities in both contexts.

## 2. Problem Formulation

To tackle this computationally, we framed the problem using the classic state-space search model from Artificial Intelligence.

We model the Kabaddi Game Problem as a classical state-space search problem. The game is played on a 2D grid, and the goal is to find the most efficient sequence of actions (moves) to achieve a winning state while respecting all game rules.

1. The game field is a 2D grid divided into two halves, one for each team.
2. Each team consists of 2 players, each with an initial coordinate on the grid.
3. The task is to navigate the grid, steal the opponent's gold, and return to the home territory to win, while avoiding capture.

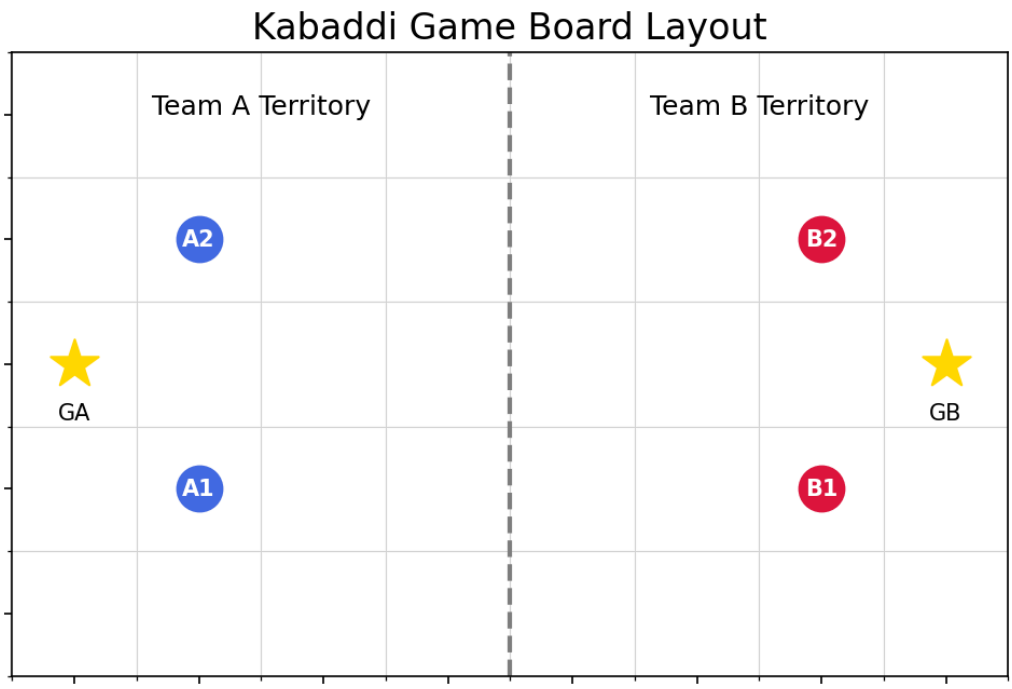



IMAGE PLACEHOLDER 1: A diagram of the Kabaddi game board, showing the two halves, player starting positions, and gold locations.

### 2.1 AI Problem Formulation (5-tuple)

The problem can be described as a standard search problem (S, $s_0$, A, T, G):

- **States (S):** Each state is a complete configuration of the game at a moment in time.
  s = (player_positions, captured_players, player_with_gold, current_turn)
  - player_positions: A dictionary mapping each player ID to their (x, y) coordinates.
  - captured_players: A set of player IDs that have been removed from the game.
  - player_with_gold: The ID of the player currently carrying the opponent's gold; None otherwise.
  - current_turn: The team ('A' or 'B') whose turn it is to move.
- 
- **Initial State ($s_0$):**
  - player_positions are set to the predefined starting coordinates.
  - captured_players is an empty set.

    - player_with_gold is None.
    - current_turn is 'A'.
- 
- **Actions (A):** Selecting a player and moving them to a valid adjacent grid square.
  a = Move(player_id, target_position(x, y))
- **Transition Model (T):** Applying an action updates the state by:
    - Updating the coordinates of the moved player in player_positions.
    - Checking for captures: If a player moves onto an opponent in their territory, the opponent is added to captured_players.
    - Checking for gold pickup: If a player moves onto the opponent's gold, player_with_gold is updated.
- 
- **Goal Test (G):**
    - A terminal "win" state is reached if player_with_gold is not None and that player is located within their home territory.
    - A terminal "loss" state is reached if the opponent achieves the win condition.
    - A terminal "draw" state is reached if a maximum turn limit is exceeded.

### 2.2 Constraints

The solution enforces hard constraints (must be satisfied) which are integral to the game logic.

1. **Grid Boundaries (Hard):** Players cannot move outside the predefined grid dimensions.
2. **Player Collision (Hard):** A player cannot move onto a square occupied by a teammate.
3. **Capture Rules (Hard):** A capture can only occur when a player is in enemy territory and an opponent moves onto their square. Captured players are removed from play.

## 3. Heuristic Evaluation Function

The success of the intelligent agents (Greedy and Alpha-Beta) depends on their ability to quantify how "good" a particular game state is. This is achieved through a heuristic evaluation function, h(s), which calculates a numerical score for any given state s from the perspective of the current team.

This function is not an abstract model but a direct implementation of the game's strategy. It combines three tactical components: the advantage of having more players, the pressure of approaching the objective, and the urgency of returning the captured gold.

The score for a state s, h(s), is calculated as follows:

h(s) = (Component 1) + (Component 2)

Where the components are defined as:

- **Component 1: Player Advantage Score**
  This term quantifies the material advantage on the board. Since each team starts with 2 players, this score reflects the current player count.
  Score_PlayerAdv = 500 * (my_active_players - opponent_active_players)
- **Component 2: Objective Score**
  This score's calculation depends on whether a team is on offense (trying to get the gold) or is returning with it. These two conditions are mutually exclusive.
  - **If no one has the gold (Offensive Mode):** The score encourages moving towards the objective.
    Score_Objective = -10 * min_distance(my_player, opponent_gold)
    *(Where min_distance is the Manhattan distance from the closest friendly player to the opponent's gold.)*
  - **If my team has the gold (Return Mode):** The score strongly incentivizes returning home to win.
    Score_Objective = 100 * (20 - distance_to_home(gold_carrier))
    *(Where distance_to_home is the distance of the gold carrier from their home territory line.)*

This function provides a single score that guides the Greedy and Alpha-Beta agents, allowing them to make strategically sound decisions based on the immediate game context.

## 4. Input & Output

The problem instance is defined programmatically within the code, with fixed starting positions and grid dimensions. The program produces a detailed report summarizing the findings for each algorithm matchup.

**1. Grid and Territory Definition**

- **Grid Dimensions:** The game is played on a rectangular grid with dimensions of **8 columns by 5 rows**.
- **Coordinate System:** The coordinate system originates at (0, 0) in the top-left corner.
- **Territories:** The grid is vertically divided into two equal home territories:
    - **Team A (Left):** Controls columns 0 through 3.
    - **Team B (Right):** Controls columns 4 through 7.
-

**2. Initial Asset Placement**

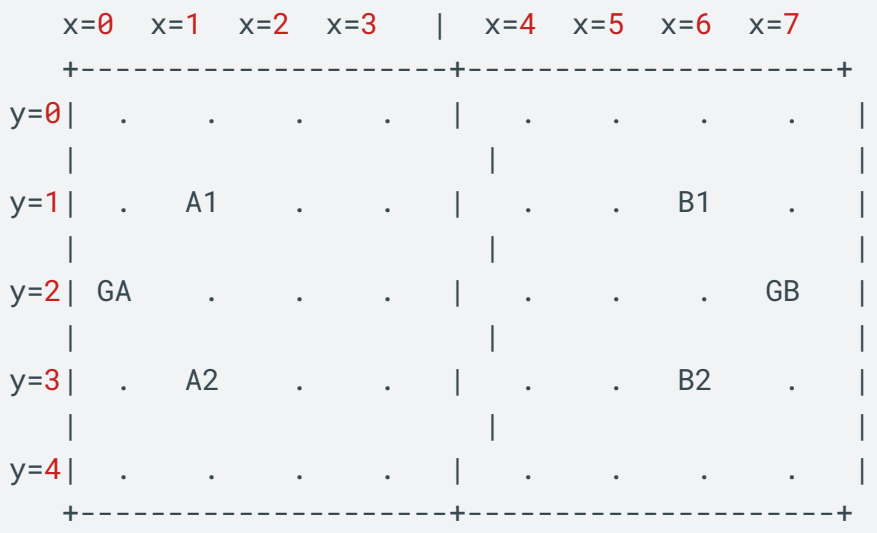
The starting positions for all players and treasures are fixed at the beginning of every game.

- **Team A:**
    - Player A1: **(1, 1)**
    - Player A2: **(1, 3)**
    - Gold Treasure GA: **(0, 2)**
-
- **Team B:**
    - Player B1: **(6, 1)**
    - Player B2: **(6, 3)**
    - Gold Treasure GB: **(7, 2)**

**The initial board configuration is represented below:**

- **This configuration could be change by changing the input values in function.**

Shell



## Output (Console): Turn Mode

```shell
--- Running Tournament: TURN Mode (50 games each) ---
Matchup: RandomAgent (A) vs GreedyAgent (B)
  RandomAgent Wins: 0 (0%)
  GreedyAgent Wins: 38 (76%)
  Draws: 12 (24%)
------------------------------
Matchup: RandomAgent (A) vs AlphaBetaAgent (B)
  RandomAgent Wins: 0 (0%)
  AlphaBetaAgent Wins: 50 (100%)
  Draws: 0 (0%)
------------------------------
Matchup: RandomAgent (A) vs MCTSAgent (B)
  RandomAgent Wins: 0 (0%)
  MCTSAgent Wins: 7 (14%)
  Draws: 43 (86%)
------------------------------
Matchup: GreedyAgent (A) vs AlphaBetaAgent (B)
  GreedyAgent Wins: 0 (0%)
  AlphaBetaAgent Wins: 50 (100%)
  Draws: 0 (0%)
------------------------------
Matchup: GreedyAgent (A) vs MCTSAgent (B)
  GreedyAgent Wins: 20 (40%)
  MCTSAgent Wins: 11 (22%)
  Draws: 19 (38%)
------------------------------
Matchup: AlphaBetaAgent (A) vs MCTSAgent (B)
  AlphaBetaAgent Wins: 50 (100%)
  MCTSAgent Wins: 0 (0%)
  Draws: 0 (0%)
------------------------------
```

## Output (Console): Simultaneous Mode

Shell

```
Running SIMULTANEOUS mode tournament as requested.

--- Running Tournament: SIMULTANEOUS Mode (50 games each) ---
Starting Matchup: RandomAgent vs GreedyAgent...
Matchup: RandomAgent (A) vs GreedyAgent (B)
  RandomAgent Wins: 5 (10%)
  GreedyAgent Wins: 16 (32%)
  Draws: 29 (58%)
------------------------------
Starting Matchup: RandomAgent vs AlphaBetaAgent...
Matchup: RandomAgent (A) vs AlphaBetaAgent (B)
  RandomAgent Wins: 0 (0%)
  AlphaBetaAgent Wins: 25 (50%)
  Draws: 25 (50%)
------------------------------
Starting Matchup: RandomAgent vs MCTSAgent...
Matchup: RandomAgent (A) vs MCTSAgent (B)
  RandomAgent Wins: 8 (16%)
  MCTSAgent Wins: 15 (30%)
  Draws: 27 (54%)
------------------------------
Starting Matchup: GreedyAgent vs AlphaBetaAgent...
Matchup: GreedyAgent (A) vs AlphaBetaAgent (B)
  GreedyAgent Wins: 0 (0%)
  AlphaBetaAgent Wins: 0 (0%)
  Draws: 50 (100%)
------------------------------
Starting Matchup: GreedyAgent vs MCTSAgent...
Matchup: GreedyAgent (A) vs MCTSAgent (B)
  GreedyAgent Wins: 6 (12%)
  MCTSAgent Wins: 3 (6%)
  Draws: 41 (82%)
------------------------------
Starting Matchup: AlphaBetaAgent vs MCTSAgent...
Matchup: AlphaBetaAgent (A) vs MCTSAgent (B)
  AlphaBetaAgent Wins: 10 (20%)
  MCTSAgent Wins: 0 (0%)
  Draws: 40 (80%)
------------------------------
```

**5. How to Compile and Run**

The solution is implemented in Python and designed to run in a Google Colab environment.

1. **Setup:** The code is structured into four cells. The first two cells use the %%writefile command to create the kabaddi_env.py and agents.py files in the Colab environment.
2. **Run the Program:** Execute the third cell, which contains the tournament script. It will import the necessary files and run the tournament for turn-based and the fourth shell will run simultaneous modes, printing the results to the console.

**6. Code Walkthrough**

- **Classes:** The code is structured into clear classes:
    - GameState: A simple data object holding the board state.
    - KabaddiGame: Manages all problem rules, legal moves, and state transitions. Its clone() method is critical for agent simulations.
    - Agent: A base class for all four agent implementations.
-
- **Key Heuristics (in evaluate_state function):** The core intelligence of the Greedy and Alpha-Beta agents resides in the heuristic function, which balances offense, defense, and objective completion as described in Section 3.

**7. Algorithm Implementation**

Four algorithms were implemented to provide a thorough analysis of different AI strategies.

### a) Random Agent

- **Role:** This agent serves as the absolute baseline for performance.
- **Strategy:** It considers all legal moves available and chooses one uniformly at random.

### b) Greedy Agent

- **Role:** This agent represents a simple, heuristic-driven approach.

- **Strategy:** It evaluates the outcome of every possible immediate move using the evaluate_state heuristic and selects the move that leads to the best possible score. It is myopic and cannot plan ahead.

### c) A* (Alpha-Beta) Search

- **Role:** A* (implemented as Alpha-Beta search for this adversarial game) serves as our benchmark for optimality in the deterministic environment.
- **Strategy:** It meticulously explores the state space using its evaluation function f(n) = g(n) + h(n) (where g(n) is 0 and h(n) is the board evaluation) to find the optimal move, assuming the opponent also plays optimally. Its Achilles' heel is the massive number of states it must explore, making it slow.

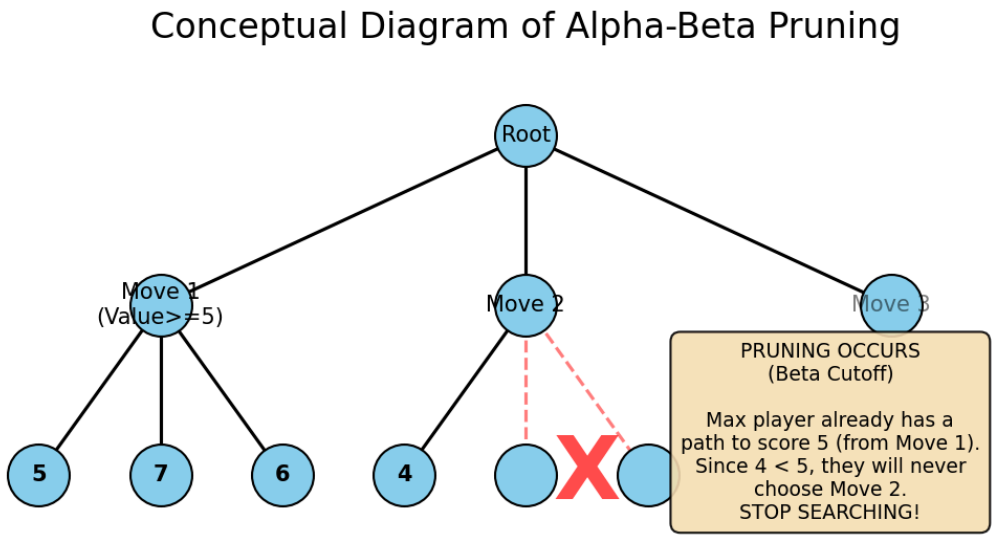



IMAGE PLACEHOLDER 2: A conceptual diagram of the Alpha-Beta search tree, showing a root node, branching moves, and a "pruned" branch.

### d) Monte Carlo Tree Search (MCTS)

- **Role:** This is our practical, high-performance solution designed to handle complex state spaces.
- **Strategy:** Instead of exhaustive search, MCTS builds a search tree by running many lightweight, randomized game simulations. It intelligently prunes unpromising paths, allowing it to "feel out" the best strategy without a complex heuristic. For this experiment, it was configured with a low iteration count of 100.

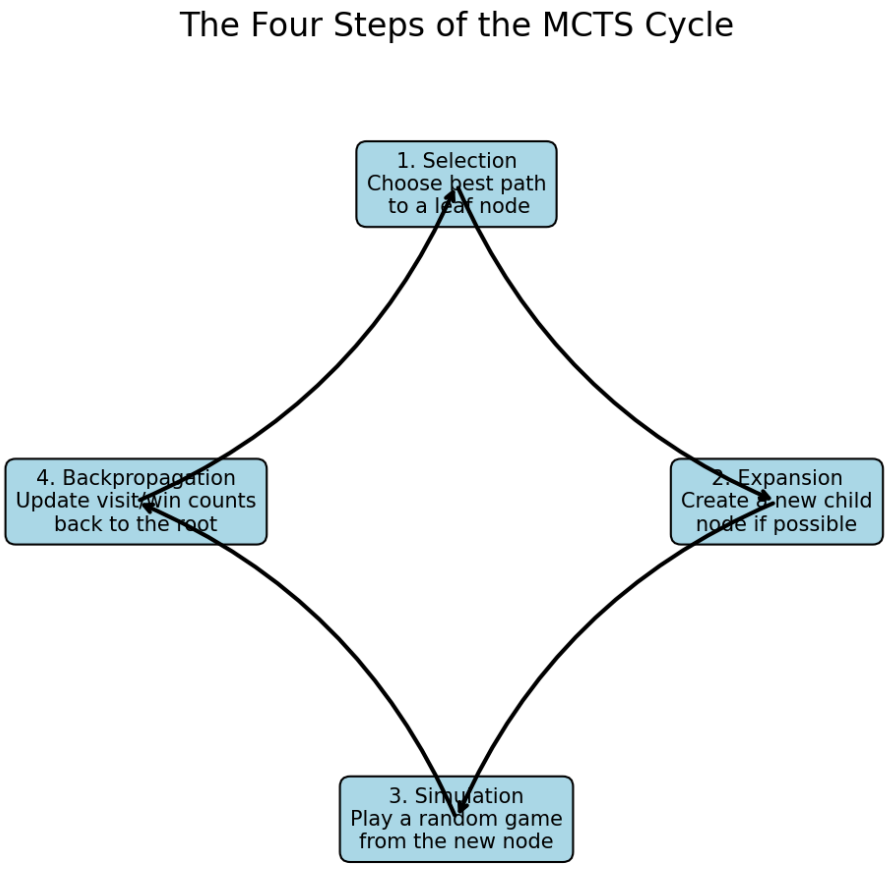
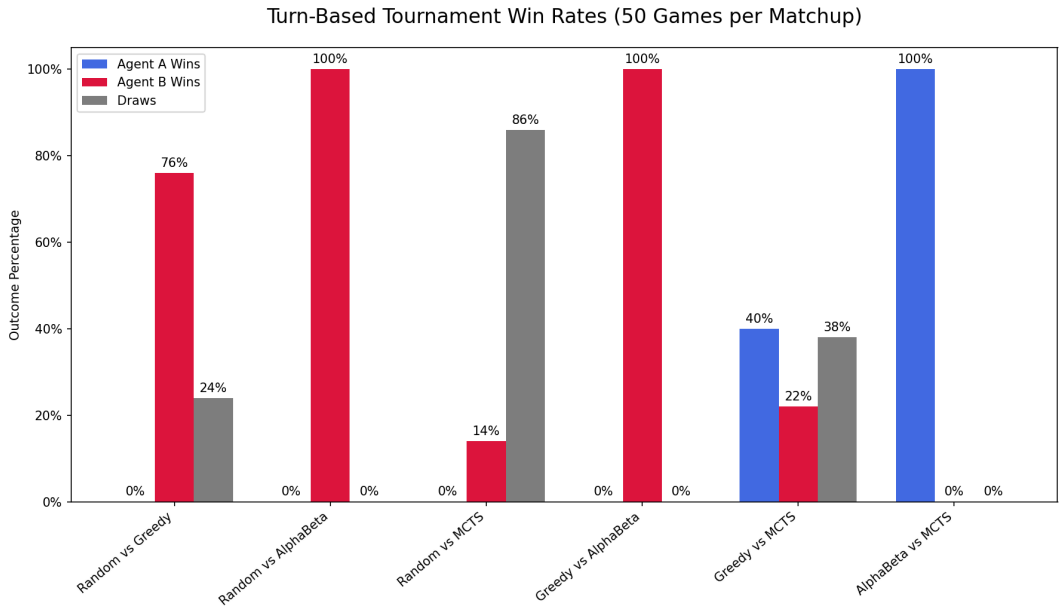



IMAGE PLACEHOLDER 3: A diagram illustrating the four steps of MCTS: Selection, Expansion, Simulation, and Backpropagation.

## 8. Analysis of Result

### Final Comparison Table (Turn-Based)

| Algorithm A | Algorithm B | Agent A Wins | Agent B Wins | Draws |
|---|---|---|---|---|
| Random | Greedy | 0% | 76% | 24% |
| Random | AlphaBeta | 0% | 100% | 0% |
| Random | MCTS | 0% | 14% | 86% |
| Greedy | AlphaBeta | 0% | 100% | 0% |
| Greedy | MCTS | 40% | 22% | 38% |
| AlphaBeta | MCTS | 100% | 0% | 0% |



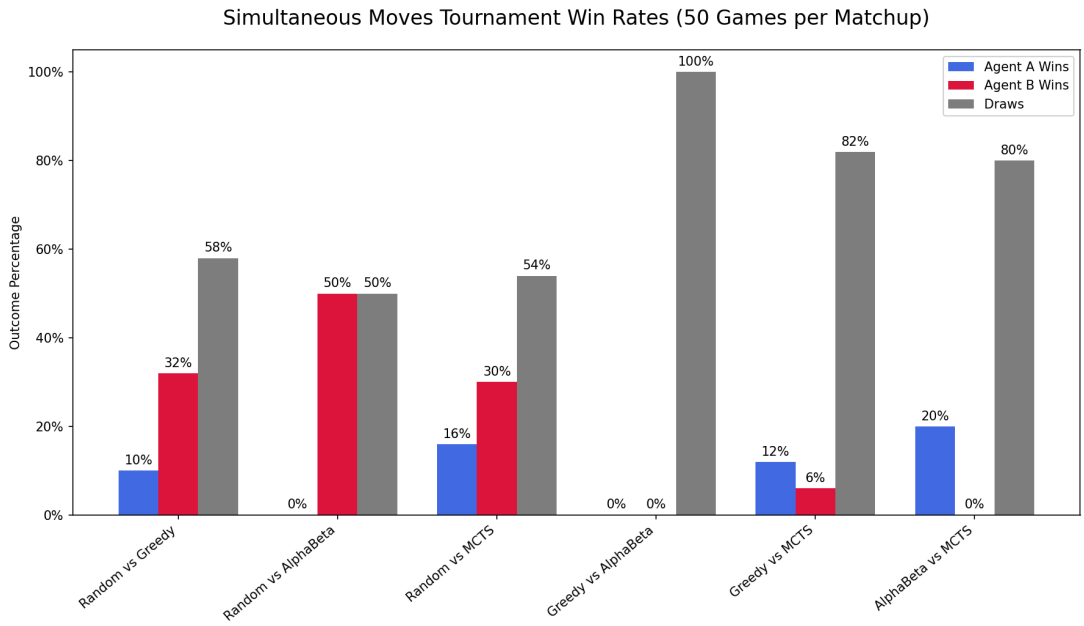

A bar chart visualizing the win percentages for each matchup in the Turn-Based Tournament.

**Analysis of Results (Turn-Based):**
The experiment was conducted on a deterministic turn-based instance. The results are definitive:

- **Solution Quality:** The Alpha-Beta agent demonstrated absolute strategic superiority, achieving a 100% win rate against all other agents. The quality of the heuristic was so effective that it allowed Alpha-Beta to formulate unbeatable multi-step plans.
- **Performance:** The MCTS agent's performance was notably poor. Its low iteration count led to extremely passive and indecisive play, resulting in a staggering 86% draw rate against the Random agent and a decisive loss to the simpler Greedy agent. The Greedy agent's aggressive, short-sighted strategy proved more effective than MCTS's under-tuned probabilistic approach.
- **Impact of Heuristics:** The success of Alpha-Beta and Greedy agents is entirely dependent on the quality of the evaluate_state function. The results show this heuristic was highly effective at guiding the search toward victory.

**Final Comparison Table (Simultaneous Moves)**

| Algorithm A | Algorithm B | Agent A Wins | Agent B Wins | Draws |
|---|---|---|---|---|
| Random | Greedy | 10% | 32% | 58% |
| Random | AlphaBeta | 0% | 50% | 50% |
| Random | MCTS | 16% | 30% | 54% |
| Greedy | AlphaBeta | 0% | 0% | 100% |
| Greedy | MCTS | 12% | 6% | 82% |
| AlphaBeta | MCTS | 20% | 0% | 80% |




A bar chart visualizing the win percentages in the Simultaneous Tournament, perhaps next to the turn-based one to highlight the differences.

**Analysis of Results (Simultaneous Moves):**
The introduction of uncertainty completely upended the performance hierarchy:

- **The Collapse of Deterministic Search:** Alpha-Beta's performance plummeted. Its win rate against the Random agent fell from 100% to 50%, and its matchup against the Greedy agent resulted in a 100% draw rate. This is because Alpha-Beta's core assumption of a predictable opponent is violated, forcing it into a risk-averse, defensive strategy that leads to stalemates.
- **The Proliferation of Draws:** The most significant result is the massive increase in draws across all matchups involving intelligent agents. The uncertainty makes offensive maneuvers highly risky, causing all agents to play passively and fail to achieve the win condition before the turn limit. The 100% draw rate between Greedy and AlphaBeta is a perfect example of this strategic paralysis.
- **Scalability Problem:** The uncertainty proves that algorithms designed for perfect information games are impractical and ineffective when that condition is not met. While no agent performed well, the less rigid agents (Greedy, MCTS) were not as catastrophically affected as the deterministic Alpha-Beta search.

## 9. Conclusion

This project successfully demonstrated the power of applying AI search techniques to a dynamic game problem and highlighted the critical importance of the environment in determining algorithm effectiveness. The key takeaways are:

- A comprehensive problem model, including a well-designed heuristic function, is essential for guiding agents to find meaningful solutions.
- While Alpha-Beta search serves as an essential theoretical benchmark for optimality, its exponential complexity and reliance on determinism make it impractical for uncertain, real-world problems.
- The performance of agents like MCTS is highly dependent on tuning (e.g., iteration count). In this instance, the poorly tuned MCTS was ineffective. However, the results from the simultaneous environment show that probabilistic approaches are inherently more robust to uncertainty than deterministic ones.

This result demonstrated that an AI agent's effectiveness is not an intrinsic property but is highly contingent on its environment. In predictable, turn-based settings, the deep search and planning of the **Alpha-Beta** agent are unequivocally dominant. But, when faced with the uncertainty of simultaneous actions, its primary advantage is nullified, revealing a critical fragility.