Kabbadi Game

AI Assignment 2 | Gour Krishna Dey | Roll - MT24035 | IIIT Delhi

Links - Github Repo for Code , Reports & PPT & Submission Drive Link

---

1. Overview

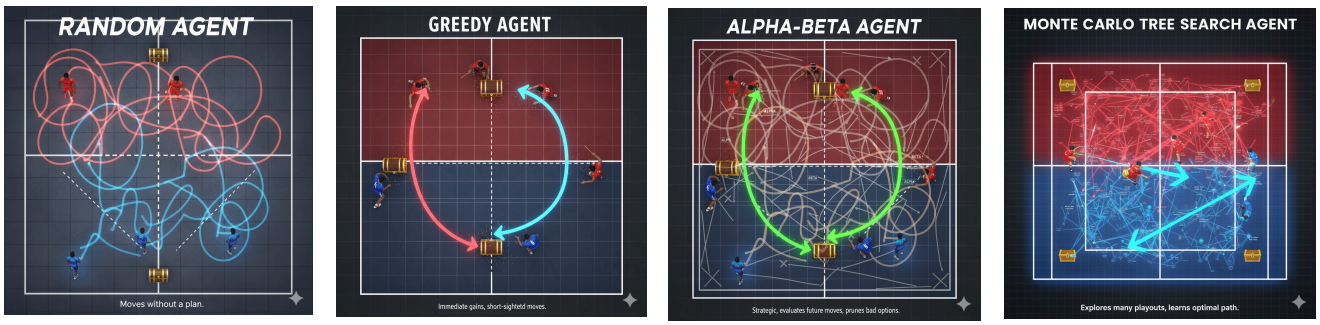
This assignment focuses on designing an **Artificial Intelligence driven Kabaddi game simulation** on a **grid based environment** . The main aim is to explore how different types of AI agents each with unique decision making logic & perform under identical game rules and environmental conditions .

In this grid based version of Kabaddi:

- There are 2 **teams (Team A & Team B)**.
- Each team has **two players** positioned on opposite halves of the grid (Their position can be given manually or default system) .
- Each team protects its **gold treasure** , which is located at a fixed position within its own half of the grid.
- A team **wins** when one of its players successfully **steals the opponent's treasure** and **returns it** to its own home territory without being captured .

The project involves:

1. Building a simulation environment that enforces all game rules.
2. Implementing four distinct agents that make move decisions using different strategies:
    - Random decision-making
    - Greedy (heuristic-based) logic
    - Alpha Beta pruning (search-based planning)
    - Monte Carlo Tree Search (simulation-based reasoning)
3. Conducting experiments where agents play against one another under different move conditions , turn based and also simultaneous for analyzing their comparative performance.



2. Problem Formulation

2.1 Game Environment

The game is played on a **2D grid** (for example, 5 × 6) divided vertically into two equal halves:

Left half → Home territory of **Team A** **&** **Right half** → Home territory of **Team B**

Each team starts with:

Two players (A1, A2 for Team A; B1, B2 for Team B).
A treasure (gold) placed within their own half.

The environment keeps track of :

Player positions and alive/dead status.
Which player is carrying treasure.
Capture events and movement legality.
Winning conditions.

Possible Moves per Player
Each player can perform one of the following actions per time-step:

1. Stay (no movement)
2. Up (move one cell upward)
3. Down (move one cell downward)
4. Left (move one cell left)
5. Right (move one cell right)

All movements are **bounded by grid limits**—players cannot step outside the field.

2.2 Game Rules

Territory Division

The grid is split into two halves: columns [0 , (cols/2 – 1)] for Team A and [cols/2 , (cols – 1)] for Team B.
Players may cross into the opponent's territory to steal treasure but risk being captured.

Capture Condition

If two opposing players occupy the same grid cell:
- The player who is in the enemy's half is captured and removed from play.
- Captured players no longer participate in future turns.

Treasure Pickup and Return

A player can pick up the opponent's treasure by stepping on its location.
The player must carry it back to their home territory (their own half).
If they succeed, their team wins immediately.

Winning and Draws

Win : A team can bring the opponent's treasure to its home side.
Draw : Both succeed simultaneously or the game exceeds the step limit (e.g., 200 turns).

2.3 Game Modes

The environment supports **two modes of play**:

1. Turn-based Mode
    - Team A moves both of its players first in a time-step.
    - Then Team B moves both of its players.
    - Captures are evaluated after each side's move.
2. Simultaneous Mode
    - Both teams move all their players **at the same time**.
    - If both achieve victory conditions simultaneously, the result is a **draw**.

2.4 Agents and Strategies

Agent Type	Description	Nature of Decision-Making
Random Agent	Selects moves completely at random for each player. Represents a baseline, non-intelligent behavior.	Non-deterministic, reactive
Greedy Agent	Chooses moves that immediately reduce the distance to the opponent's treasure (Manhattan distance heuristic). Ignores future consequences or traps.	Heuristic, short-sighted
Alpha-Beta Agent	Uses minimax search with alpha-beta pruning to evaluate possible move sequences. Considers opponent responses up to a certain search depth and chooses moves maximizing long-term gain.	Strategic, look-ahead
MCTS Agent	Uses Monte Carlo Tree Search: runs multiple random simulations ("rollouts") from each possible move and selects the move with the highest simulated win rate.	Statistical, adaptive

Each agent uses the same environment and move set but differs in how it decides the next action, allowing us to compare pure chance vs heuristic vs search-based vs probabilistic planning.

2.5 Objective of the Simulation

The goal is to build a common environment where all agents interact under identical rules, run multiple matches between every agent pair, record outcomes such as wins, losses, draws, average steps, and captures, and analyze how different decision-making strategies perform in an adversarial multi-agent setting.

3. Implementation Overview

Before coding, the entire problem was broken down conceptually:

1. *Environment Design* — to simulate a grid-based game world (positions, moves, treasure, capture, etc.).
2. *Player Modeling* — each player needed a position, status (alive/dead), and treasure-carrying flag.
3. *Rule Engine* — logic to check captures, treasure pickup, and winning conditions.
4. *Agent Strategie*s — four distinct AI behaviors to test different reasoning methods.
5. *Game Execution* — to handle movement order, turn modes, and environment updates.
6. *Analysis Framework* — tournament simulation to evaluate agent performance over multiple games.

This systematic breakdown ensured that every component had a specific role, which also made debugging and extending the simulation easier.Here are the code summary

3.1. Position and Player Structures

- *Position*: Represents grid coordinates (x, y) and allows position comparison.
- *Player*: Stores player's position, alive status, and treasure possession state.
  Used for tracking each agent's movement and game condition.

3.2. GameState Class

This class defines the core game environment and rules.

Key Attributes:

- *rows*, *cols*: Grid dimensions.
- *teamA*, *teamB*: Players of each team.
- treasureA, treasureB: Treasures for both teams.
- turnBased: Defines gameplay mode (sequential/simultaneous).
- capturesA, capturesB: Count of captures made.

Important Methods:

- *applyMoves()* → Updates positions of all players as per agent decisions.
- *checkCaptures()* → Implements capture rules when players collide in enemy half.
- *checkTreasurePickup()* → Detects treasure collection.
- *checkWin()* → Checks if a team successfully brings the treasure home.
- *printBoard()* → Displays the current board state visually.

3.3. Agent Base Class

Abstract parent class defining a **common interface**:

```
vector<int> getMoves(const GameState &state, int teamId);
```

All intelligent agents inherit and override this to decide their own move strategies.

3.4. Agent Types and Logic

- RandomAgent → Pick moves randomly.
- GreedyAgent → Moves directly toward the opponent's treasure using distance heuristic.
- AlphaBetaAgent → Uses Minimax with Alpha–Beta pruning to plan ahead strategically.
- MCTSAgent → Uses Monte Carlo simulations to estimate the best move statistically.

3.5. Evaluation Function

- **evaluateState()** → Scores a game state based on:
    - Win/loss conditions
    - Treasure possession
    - Capture counts
    - Distance to target treasure
      Used by Alpha–Beta agent to guide search decisions.

3.6. Simulation Functions

- *playGame()* → Simulates one complete game between two agents and returns results (winner, steps, captures).
- *simulateTournament()* → Runs multiple games among all agent pairs and records statistics for comparison.

3.7. main() Function

- Handles user input for custom or default setup.
- Initializes environment and agents.
- Allows choosing between:
    - **Single Game Mode** (manual play between chosen agents).
    - **Tournament Mode** (automated simulations across all agents).
- Displays summarized results with win counts and performance stats.

5. Input & Output

In our set up we can simulate two types of game one is Individual game another is Tournament .

Game between Two Individual Agents (Ex - Random vs Greedy) : Here I am showing a game simulation between two agents one is Random and Other is Greedy for Turnbased movement. Similarly we can run this with Simultaneous moves also . With user input we can run simulation with default set up or any particular setup also .

```
Shell
PS E:\MTechCSE\Study\Sem3\AI\Assignment\Assignment_2> ./a
Do you want default setup? (y/n): y
Using default setup (5x6).

Choose mode: 1 = Single Game, 2 = Tournament: 1
Choose Agent for Team A (1=Random,2=Greedy,3=AlphaBeta,4=MCTS): 1
Choose Agent for Team B (1=Random,2=Greedy,3=AlphaBeta,4=MCTS): 2
Choose gameplay type: 1=Turn-based,2=Simultaneous: 1
Game Start: Team A(Random) vs Team B(Greedy)
```

```
A . . | . . .
A . . | . . .
. T . | . t .
. . . | . . B
. . . | . . B
----------------------
Step 1:
Team A moves: Right Right
Team B moves: Left Left
.......
----------------------
Step 2:
----------------------
........
Step 49:
Team A moves: Left Left
Team B moves: Stay Stay
. . . | . . .
. . . | . . .
A T A | . t .
. . . | . . .
. . . | . . .
----------------------
Winner: Team A
Steps: 49 Captures(A:2, B:0)
```

Game between All Individual Agents (A Tournament like setup) : Here I am showing a tournament simulation between multiple agents for Turnbased movement . Similarly we can run this with Simultaneous moves also . With user input we can run simulation with default set up or any particular setup too .

Shell

```
PS E:\MTechCSE\Study\Sem3\AI\Assignment\Assignment_2> ./a
Do you want default setup? (y/n): y
Using default setup (5x6 board).

Choose mode: 1 = Single Game, 2 = Tournament: 2
Choose gameplay type for tournament: 1=Turn-based, 2=Simultaneous: 1
Enter number of games per pairing: 100
Show concise per-game lines? (y/n): n

===== Tournament: 100 games per ordered pair =====
Random (A) vs Greedy (B) --> A_wins=99, B_wins=0, Draws=1, avgStepsA=75.60
Random (A) vs AlphaBeta (B) --> A_wins=42, B_wins=51, Draws=7, avgStepsA=88.48, avgStepsB=81.92
Random (A) vs MCTS (B) --> A_wins=44, B_wins=49, Draws=7, avgStepsA=85.45, avgStepsB=97.61
Greedy (A) vs Random (B) --> A_wins=0, B_wins=98, Draws=2, avgStepsB=68.78
Greedy (A) vs AlphaBeta (B) --> A_wins=0, B_wins=100, Draws=0, avgStepsB=72.45
Greedy (A) vs MCTS (B) --> A_wins=0, B_wins=99, Draws=1, avgStepsB=68.19
AlphaBeta (A) vs Random (B) --> A_wins=47, B_wins=37, Draws=16, avgStepsA=91.57,
avgStepsB=94.03
AlphaBeta (A) vs Greedy (B) --> A_wins=100, B_wins=0, Draws=0, avgStepsA=73.08
AlphaBeta (A) vs MCTS (B) --> A_wins=50, B_wins=47, Draws=3, avgStepsA=94.58, avgStepsB=90.30
MCTS (A) vs Random (B) --> A_wins=44, B_wins=47, Draws=9, avgStepsA=90.30, avgStepsB=79.49
MCTS (A) vs Greedy (B) --> A_wins=99, B_wins=0, Draws=1, avgStepsA=77.87
MCTS (A) vs AlphaBeta (B) --> A_wins=51, B_wins=41, Draws=8, avgStepsA=87.94, avgStepsB=90.44
```

7. Interpretation of Results Analysis

7.1. Individual Game Analysis

In the single-game test case:

```
Shell
Team A: Random Agent
Team B: Greedy Agent
Gameplay: Turn-based
Result: Team A (Random) wins in 49 steps with 2 captures.
```

Interpretation:

- Even though the Greedy Agent moves directly toward the opponent's treasure, its strategy lacks adaptability.
- The Random Agent occasionally benefits from unpredictable movement, which can confuse a deterministic opponent like Greedy.
- The game lasted 49 steps — indicating moderate interaction before conclusion.
- Team A successfully captured two opponents, suggesting that the Greedy strategy exposes players to higher risk during pursuit.

Thus, the Random Agent's unpredictability occasionally outperforms simple heuristics in small environments where collisions are frequent.

7.2. Tournament Analysis

Turn-Based Mode (Sequential Moves)

Agent Matchup	Summary Observation
Random vs Greedy	Random dominates (99 wins), showing Greedy's predictability.
Random vs AlphaBeta / MCTS	Near balanced outcomes , strategic agents adapt to randomness.
Greedy vs All	Performs worse , zero wins against all others.
AlphaBeta vs MCTS	Very close results (50 vs 47), showing both perform comparably in structured play.
Overall	AlphaBeta and MCTS emerge as consistently strong; Greedy fails in all cases.

Key Insights:

- **AlphaBeta** achieves the highest consistency due to its look-ahead mechanism.
- **MCTS** performs similarly but varies slightly due to probabilistic rollouts.
- **Greedy Agent** fails entirely in competitive setups — it lacks future planning.
- Average steps per win (70–90) indicate that turn-based play encourages deeper strategy rather than fast captures.

Simultaneous Mode (Concurrent Moves)

Agent Matchup	Summary Observation
Random vs Greedy	Random again dominates.
Random vs AlphaBeta / MCTS	Slightly more balanced than turn-based, due to simultaneous move uncertainty.
AlphaBeta vs MCTS	AlphaBeta wins more frequently (59 vs 34), showing stronger planning in parallel dynamics.
Overall	**AlphaBeta outperforms all**, especially when both sides move together, showing its ability to anticipate uncertainty better than MCTS.

Key Insights:

- Simultaneous play introduces greater unpredictability; random noise affects deterministic agents.
- AlphaBeta adapts better, leveraging depth-based search to minimize risk.
- MCTS still performs well but less consistently due to stochastic simulations.
- Greedy Agent again remains weakest.

8. Final Comparison

Mode	Best Performing Agent(s)	Key Characteristics	Average Behavior
Turn-Based	AlphaBeta ≈ MCTS	Balanced between search and simulation; strong long-term reasoning.	More deterministic outcomes; longer games (~80 steps).
Simultaneous	AlphaBeta	Handles uncertainty better, adapts to concurrent movement.	More dynamic and shorter matches (~70 steps).
Overall Weakest	Greedy Agent	Too rigid; fails against adaptive agents.	Loses in all pairings.
Most Unpredictable	Random Agent	Can occasionally outperform heuristics.	Highly inconsistent.

9. Conclusion

From the analysis, several key insights emerge about the performance of different agents under various game modes. In turn-based games, AlphaBeta and MCTS perform similarly, demonstrating a strong balance between deep search and simulation, resulting in more deterministic outcomes and longer games. In simultaneous-move scenarios, AlphaBeta adapts better to uncertainty and concurrent actions, producing faster and more dynamic matches. The Greedy Agent consistently underperforms, showing rigidity and inability to cope with adaptive strategies. Interestingly, the Random Agent, while highly inconsistent, can occasionally surpass heuristic-based agents, highlighting the potential value of unpredictability in certain situations. Overall, agent effectiveness depends strongly on both the game mode and the ability to balance strategy with adaptability.

The study shows that sophisticated search-based agents like AlphaBeta and MCTS generally outperform simpler or purely random strategies, but their relative advantage varies with game dynamics. While deterministic approaches excel in structured, turn-based settings, adaptability and handling uncertainty are crucial in simultaneous-move games. Greedy strategies are too rigid to compete effectively, whereas occasional randomness can sometimes yield surprising results. Designing agents requires a careful balance between strategic depth, flexibility, and controlled unpredictability to maximize performance across different scenarios.

---------- X ----------