Kabbadi Game

CSE 643 - HW ASSIGNMENT 2 (SEC A)

Video Presentation Link

Anika Aharwal

2023086

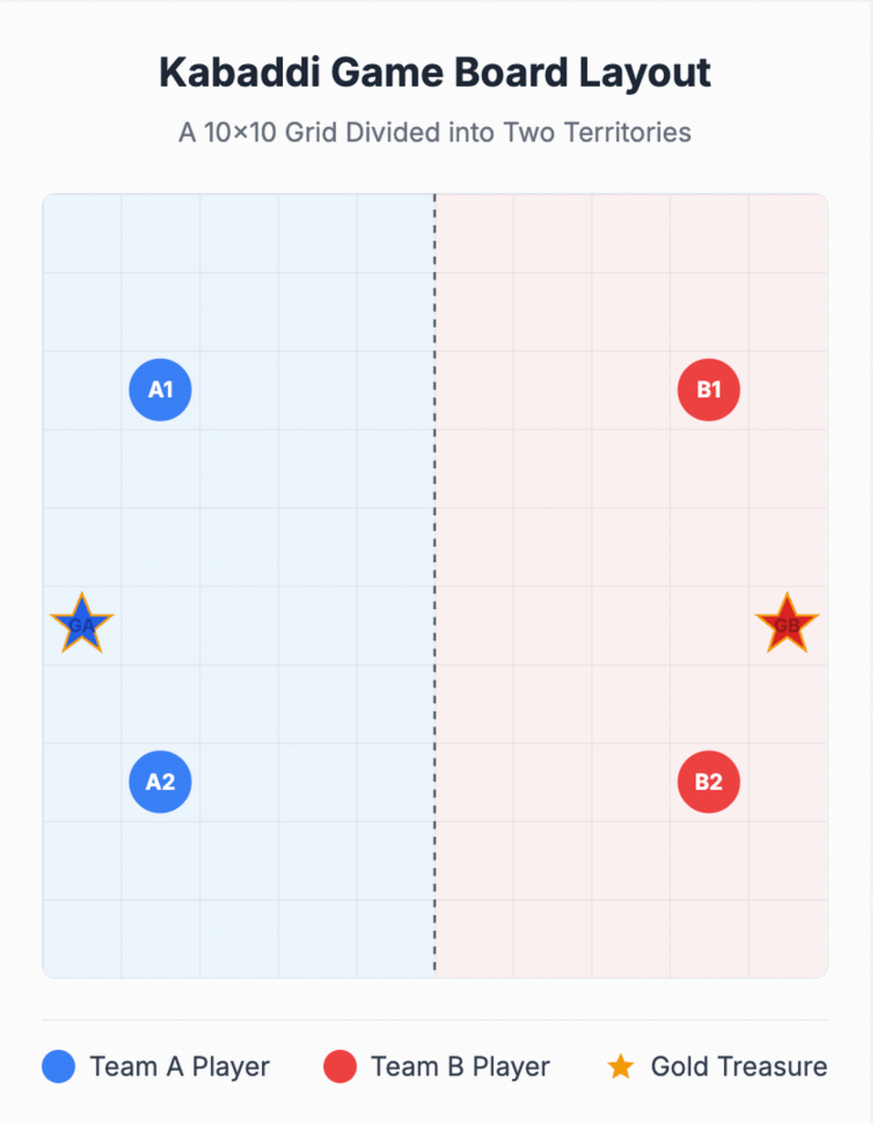
Problem Formulation

1. **Model the Environment:** Simulate a 2v2 Kabaddi game with turn-by-turn and simultaneous game play modes
2. **Design Four AI Agents:** Implement Random, Greedy, Alpha-Beta, and MCTS agents.
3. **Analyze Agent Performance:** Run tournaments to gather data and draw conclusions about the most effective strategies for each environment.

Game Environment

Key Rules of the Game:

- **Objective:** Steal the opponenet’s gold and return home
- **Movements:** Move one or both players one square per turn
- **Capture:** Occurs on an opponent’s square in their territory
- **Game Modes:**
  - Turn-by-turn
  - Simultaneous



Heuristic Function Design

Purpose: Estimate how favorable a game state is for our team. Higher score → stronger position.

$$S(state) = \Sigma\ Score(p)$$

$$Score(p) = -1000 \cdot I(is\_captured(p)) + 500 \cdot I(has\_gold(p)) - 10 \cdot d_{home}(p) \cdot I(has\_gold(p)) - 5 \cdot d_{gold}(p) \cdot (1 - I(has\_gold(p))) + \Sigma\ (d_{defence}(p,o) - d_{threat}(p,o))$$

This heuristic balances **offense, defense, and survival by:**

- Penalizing risky or captured players
- Rewarding successful raids
- Encouraging safe returns
- Adapting positioning based on opponent proximity

Term	Meaning	Purpose
−1000 × I(is_captured)	Survival penalty	Avoid being captured
+500 × I(has_gold)	Primary objective	Encourage capturing gold
−10 × d_home × I(has_gold)	Return incentive	Promotes moving gold to home
−5 × d_gold	Offensive push	Move toward enemy gold
Σ(d_defense − d_threat)	Defense & safety	Balance attack and defense

Agent Design

- **Random:** For each (non-captured) player, picks a random valid move.
  - **Implementation:** The agent gets a list of valid moves from the environment and choosed one at random for each player.
- **Greedy**: Selects the action that maximizes the immediate heuristic score, without looking at future turns.
  - **Implementation:** Uses a heuristic function that calculates the score based on factors like player safety and proximity to the gold.
- **Alpha-Beta:** An adversarial search algorithm that explores a game tree to a fixed depth and finds the optimal move by exploring a game tree, pruning branches that won't affect the outcome.
  - **Implementation:** Searches to a fixed depth using the same heuristic used in greedy. For simultaenous game mode, two versions of Alpha-Beta is used:
    - Naive Alpha-Beta
    - Fallback Alpha-Beta
- **MCTS:** A probabilistic search algorithm that uses random simulations (rollouts) to estimate the value of moves.

Results

```
--- Tournament Results: TURN_BY_TURN Mode (20 games per matchup) ---

WIN COUNT (Row player vs Column player):
                  Random  Greedy  AlphaBeta (d=2)  MCTS (t=0.5s)
Random                 0       0                0              0
Greedy                19       0                0             14
AlphaBeta (d=2)       16       0                0              5
MCTS (t=0.5s)          4       0                0              0

OVERALL SUMMARY:
                  Total Wins  Total Losses  Total Draws  Win %
Greedy                    33             0           27   55.0
AlphaBeta (d=2)           21             0           39   35.0
MCTS (t=0.5s)              4            19           43    6.1
Random                     0            39           17    0.0
------------------------------------------------
```

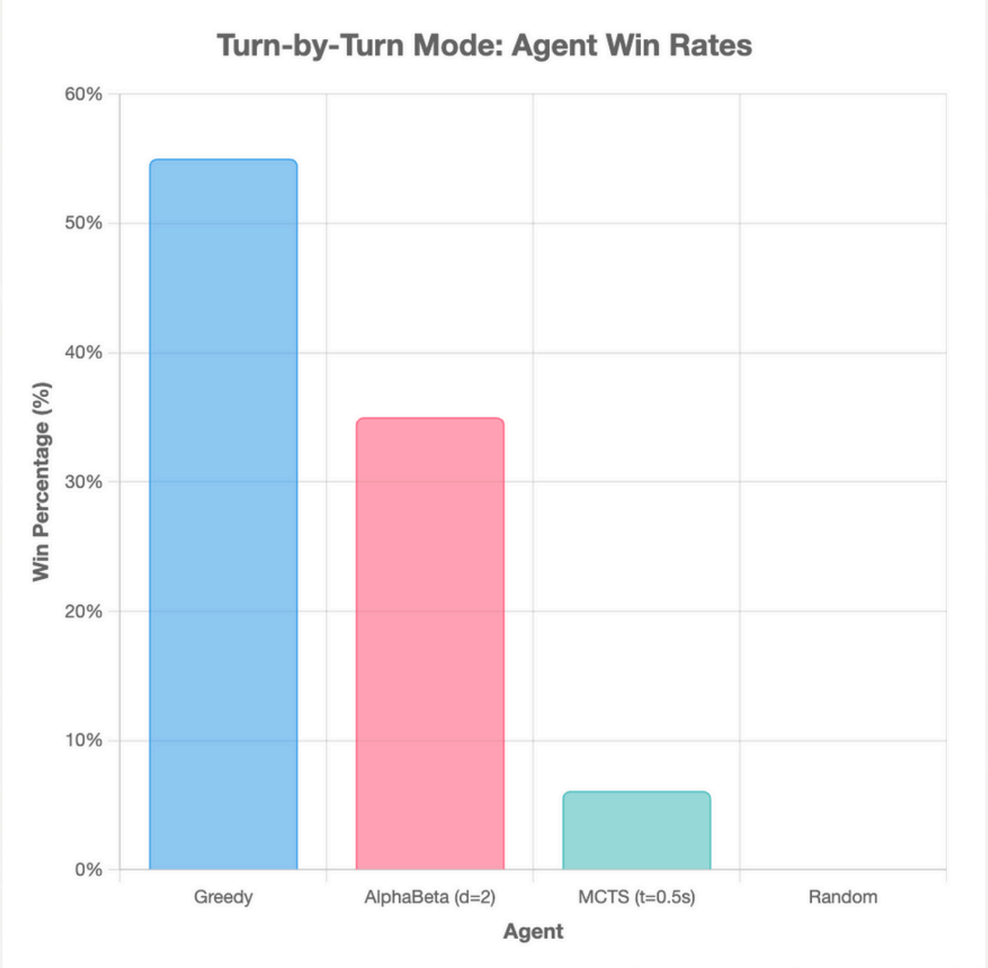
```
--- Tournament Results: SIMULTANEOUS Mode (20 games per matchup) ---

WIN COUNT (Row player vs Column player):
                    Random  Greedy  AlphaBeta (Naive)  AlphaBeta (Fallback)  MCTS (t=0.5s)
Random                   0       0                  0                     0              1
Greedy                  20       0                  0                     0             15
AlphaBeta (Naive)       19       0                  0                     0              8
AlphaBeta (Fallback)    20       0                  0                     0             14
MCTS (t=0.5s)           16       0                  5                     2              0

OVERALL SUMMARY:
                    Total Wins  Total Losses  Total Draws  Win %
Greedy                      35             0           43   44.9
AlphaBeta (Fallback)        34             2           44   42.5
AlphaBeta (Naive)           27             5           48   33.8
MCTS (t=0.5s)               23            38           15   30.3
Random                       1            75            3    1.3
------------------------------------------------
```

Analysis: Turn-by-Turn Mode

- Greedy agent's 55% win rate indicates its heuristic function is highly effective in this setting.
- The Alpha-Beta agent performed below the Greedy agent. A search depth of 2 was not sufficient to consistently find plans that outperformed the Greedy agent's immediate moves.
- The MCTS agent's 6.1% win rate highlights its sensitivity to the time limit. The number of simulations was insufficient to build an accurate value estimation for moves.
  - Hence with a tight time limit, its search was less effective than a strong, direct heuristic.



Analysis: Simultaneous Mode

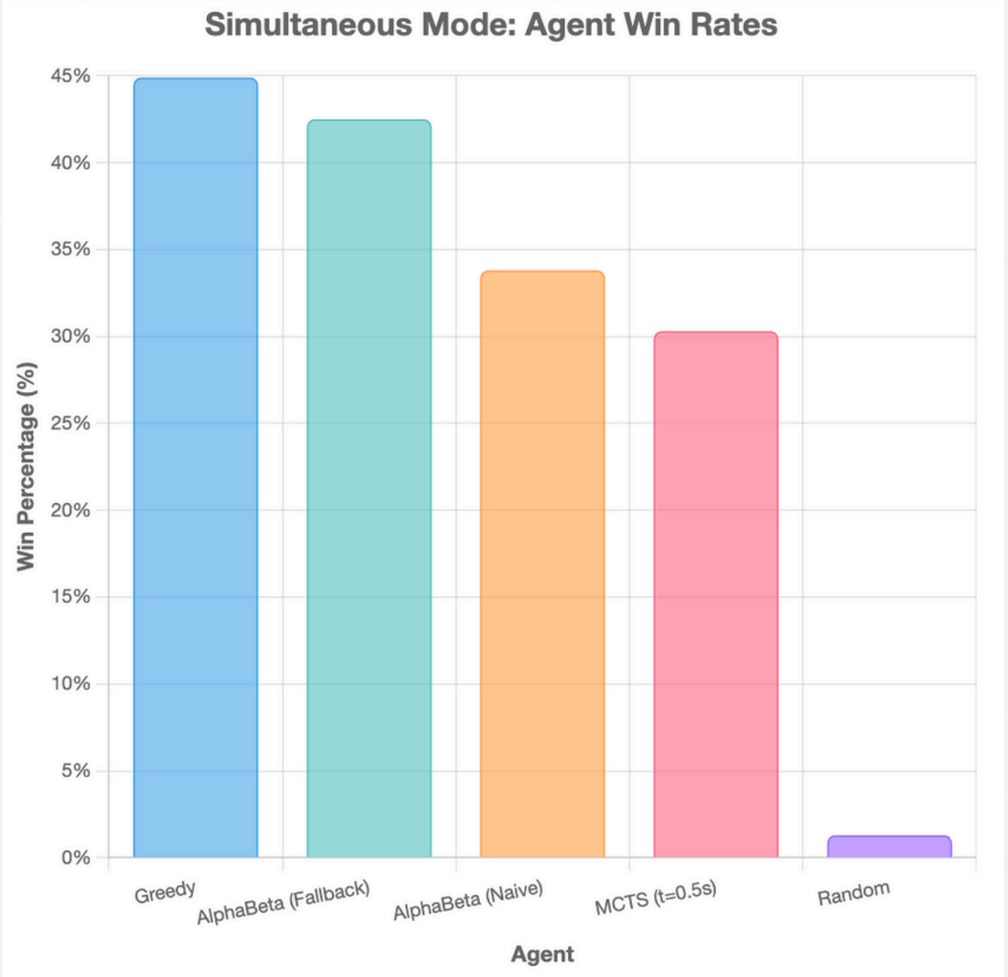
Alpha-Beta Variants

- **Problem:** Simultaneous mode breaks Alpha-Beta's turn-based assumption.
- **Agents Tested:**
  - **Naive:** Ignores the violation.
  - **Fallback:** Adapts with a Greedy strategy.
- **Results:**
  - **Fallback:** 42.5% win rate
  - **Naive:** 33.8% win rate

→ Adaptive strategies handle uncertainty better.

Overall Analysis:

- Greedy and Fallback agents performance confirms that reactive policies are effective in environments with high uncertainty.
- MCTS improved (30.3% win rate) - its probabilistic model fits simultaneous environments better than Naive Alpha-Beta's deterministic model.



Conclusion

- **Environment Dictates Strategy:** The optimal agent strategy is dependent on the environment's properties.
- **Adaptive Design:** The comparison of Alpha-Beta variants provided evidence that an agent engineered to adapt its strategy is more robust than a rigid one.
- **Heuristics vs. Search Time:** The results show a trade-off. A strong heuristic can be more effective than a search algorithm when computational time is limited.