**Project Title:**

**Dice Wars with AI Using Monte Carlo Tree Search**

**Submitted By:**

Ammar Shah 22k-4386

Taha walaqadr 22k-4312

Muhammad Ahsan 22k-4271

**Course:**

AI

**Instructor:**

Abdullah Yaqoob

**Submission Date:**

11-5-2025

**1. Executive Summary**

**Project Overview:**  
This project is a Python-based implementation of a 2-player grid-based strategy game modeled after Dice Wars. It involves a human player competing against an AI powered by Monte Carlo Tree Search (MCTS). The original game concept was adapted to support a flexible grid size (default 4×4) and simulate dice-based territorial battles. The AI uses MCTS to simulate outcomes, evaluate options, and make decisions within a time or simulation limit.

**2. Introduction**

**Background:**

Dice Wars is a turn-based strategy game that revolves around area control using dice rolls to attack and defend territories. The original game concept focuses on multiple players competing to conquer the board by occupying all the regions. This project recreates the game using Python, adapting it to a two-player format: one human and one AI, to explore AI decision-making using simulation-based strategies.

**Objectives of the Project:**

* Implement a simplified version of Dice Wars in Python.
* Design a working MCTS algorithm capable of choosing optimal moves.
* Allow human-vs-AI interaction with varying board sizes.
* Evaluate the performance of the AI based on simulation count and decision quality.

**3. Game Description**

**Original Game Rules:**

* Each cell on the grid represents a territory and contains a number of dice.
* Players can attack neighboring territories only if they have at least 2 dice.
* A dice roll (random 1–6 for each die) decides the result of the attack.
* The game ends when one player owns all territories.

**Innovations and Modifications:**

* Supports custom board sizes (default 4×4).
* Added MCTS-based AI for simulating and choosing moves.
* Integrated win condition logic for stalemates and control dominance.
* Dynamic adjustment of AI depth based on board size.

**4. AI Approach and Methodology**

**AI Techniques Used:**

* **Monte Carlo Tree Search (MCTS)**: An iterative simulation-based search algorithm that balances exploration and exploitation using the UCB1 formula.

**Algorithm and Heuristic Design:**

* The AI clones the current game state, simulates hundreds of games from each possible move, and backpropagates the outcomes.
* **Selection**: Chooses the child node with the highest UCB1 score.
* **Expansion**: Adds a new move node if not already expanded.
* **Simulation**: Randomly plays out a game to the end from that state.
* **Backpropagation**: Updates statistics (wins and visits) up the tree.
* A max\_steps cap is enforced to prevent infinite simulations.

**AI Performance Evaluation:**

* Decision speed was under 5 seconds for a 4x4 grid with ~75 simulations.
* The AI was consistently able to win or tie based on smart simulations.
* More simulations resulted in better decision quality but longer delays.

**5. Game Mechanics and Rules**

**Modified Game Rules:**

* Players can attack only adjacent enemy cells and must have ≥2 dice.
* Attack success is based on dice roll totals.
* A losing attacker is reduced to 1 die; a winning attacker conquers.

**Turn-based Mechanics:**

* Human and AI take turns.
* If a player has no valid moves, the turn is skipped.
* If both players have no moves, the winner is decided by territory count.

**Winning Conditions:**

* A player wins by owning all cells.
* If both players are stuck, the player with more cells wins.
* In case of a tie in territory count, a winner is chosen randomly.

**6. Implementation and Development**

**Development Process:**

* Designed and tested game mechanics using object-oriented Python.
* Implemented MCTS from scratch with dynamic simulation depth.
* Integrated human input parsing and win condition logic.

**Programming Languages and Tools:**

* **Language**: Python
* **Libraries**: random, math, copy, time
* **Tools**: Visual Studio Code / Jupyter Notebook (for development and testing)

**Challenges Encountered:**

* Preventing infinite simulation loops in MCTS.
* Balancing performance and decision accuracy.
* Designing a fair resolution in stalemate situations.

**7. Team Contributions**

* **Ammar:** Developed the expansion and simulation part of the MCTS algorithm
* **Taha:** Developed the backpropogation part of the MCTS algorithm
* **Ahsan:** Developed the structure of the grid and created the main game loop

**8. Results and Discussion**

**AI Performance:**

* AI consistently made good decisions under time constraints.
* Simulation limit scaling with board size helped optimize run-time.
* Win rates ranged depending on simulation count.

| **Board Size** | **Simulations** | **Avg. Decision Time** | **Win Rate (AI)** |
| --- | --- | --- | --- |
| 4×4 | 75 | ~3–4 sec | ~85–75% |
| 5×5 | 50 | ~3–5 sec | ~70–60% |

**9. References**

* Dice Wars Game Mechanics from google
* Monte Carlo Tree Search from youtube videos and GeekforGeeks