University of Puerto Rico
Mayagüez Campus
Mayagüez, Puerto Rico

Department of Electrical and Computer Engineering

Adversarial Search

by:
Manuel Alejandro Umaña Rodriguez
Dahyna Gabrielle Martínez Pérez
Jan Luis Pérez de Jesús
Christopher Hans Mayens Matías

For: Profesor José Fernando Vega Riveros
Course: ICOM 5015, Section 001D
Date: April 23, 2025

Abstract:

This report explores the implementation of a Dominoes game utilizing adversarial search through Monte Carlo simulation. The objective was to develop a game with several different configuration options and to evaluate the performance of the AI agent. It was decided to use a Monte Carlo simulation for this agent due to its simplicity and effectiveness in uncertain and changing environments. The simulation will run 30 times per turn as it was found that more than 30 simulations per round significantly affected the computational costs of the game especially when more than one AI agent was present in the game mode. The AI simulates potential outcomes and chooses the move that will increase the chances of achieving victory by simulating the entire game. By doing this it is able to account for possible moves the player will make by estimating the tiles the other players have in their possession. The performance would be influenced directly by the number of simulations, as the AI would be able to have higher probability of being able to predict what the other players will do, and the initial tile distribution of the game. To obtain a robust sample that would reflect the capabilities of the AI several tests were performed including 2 player, 4 player, free for all, and team based simulations all using the best of 3 games to give the AI some opportunity to demonstrate its capabilities regardless of its initial tiles. As the initial tile distribution is dependent on the random nature of the game. During the project it was found that different amounts of simulations resulted in different efficiencies of the AI agent. Having a large number of simulations resulted in a more optimal solution. With the AI agent being able to force the game into a stalemate or draw consistently if it could not outright win the game, with it being able to counteract human players' suboptimal choices. However, with the increase in simulations came at the cost of computational time, with the AI player taking a significant amount of time to complete all the different simulations. Taking this into consideration, the choice for a slow but effective game with a high number of simulations or a faster but less efficient agent has to be made. For the purpose of this project it was decided to focus more on a faster game at the cost of efficiency as it would not be a good game if the AI agent slowed all other parts of the game.

Table of Contents:

Group Collaboration:

The tasks amongst each team member were divided by each member's experience utilizing python, as follows:

-Jan Pérez was in charge of creating the game modes for 1 player and 1 AI agent, 1 player and 3 AI agents,2 players and 2 AI agents, the different layout for multiplayer; to choose teams, and making the pass screen that appears when passing the computer for other players for multiplayer.

-Dahyna Martínez was in charge of creating the game modes 3 players and 1 AI, 2 AI players no human, 4 AI spectating mode as well as the scrolling bar and buttons, and making domino pieces vertical and horizontal for the game.

-Manuel Umaña was in charge of the creation of the menu for the game, making the team options for the multiplayer setting for the domino game, fixed the implementation of the pass screen when in local multiplayer; as well as implementing the highest double piece rule which decided the players turns in the game.

-Christopher Mayens was in charge of developing the performance measurement for all the files in the domino game to find the efficiency of the AI component.

-All members of the group were included and participated in the video. Video editing was made by Jan Pérez.

I. Introduction:

This project focuses on the development of a Puerto Rican Domino game that features multiple modes of play involving both human and AI players. Dominoes is a tile-based strategy game where players take turns placing tiles with matching numbers end to end, aiming to be the first to play all their tiles. The main goal of this implementation was to test whether an AI agent could efficiently play the game using traditional rules while functioning in various configurations alongside or against human players. To guide the AI's decision-making, the Monte Carlo simulation technique was used, as it allows the agent to simulate numerous possible game outcomes and make informed moves based on statistical results. Compared to other adversarial search techniques such as stochastic games and partially observable game models, which require complex representations of uncertainty or opponent behavior, Monte Carlo simulation offers a simpler yet effective approach for a game like Dominos. Another objective of the project was to provide users with an interactive experience where they could play both alongside and against AI players across a variety of game setups. The game includes 2-player and 4-player modes, each with different variations that support various combinations of human and AI participants, including both individual and team-based play. This report explains how the game was implemented, how the AI strategy works, the available game modes, and how players interact with the system.

II. Related Work:

Prior to the development of the programming project, it was determined that the environment would be partially observable, discrete, stochastic and strategic, static, sequential, multi-agent and it required a search algorithm. For this assignment the Monte Carlo simulation was selected as it offered a simple implementation that had a decent level of complexity.

Monte Carlo simulation is a technique used to estimate the behaviour of complex systems through random sampling [1]. It strongly relies on randomness which gives it a level of prediction that straight forward simulations don’t possess. This method is widely utilized in different fields such as finance, engineering, and artificial intelligence for these characteristics.

The mathematical expression used for describing this simulation is the following:

$$E[f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

where:
$f(x)$ is the value of a function over a domain D,

N is the number of random samples
x_i is the random samples from the domain D

The rules for a standard game of Domino are the following:

Before each game, players shuffle the tiles face down, and the shuffler draws last. The first play is determined by the player holding the highest double tile. Players then draw their hands, and any leftover tiles remain face down for drawing if needed, within a stock. The first player sets a tile to start the game, and play proceeds clockwise. Players match tiles to the open ends, with doubles played crosswise. If a player can't play, they must pass or draw from the stock. Game ends when a player runs out of tiles or no moves remain, with scoring based on the sum of the numbers on the unplayed tiles in all players' hands. In team play, the winning team earns the total points from all players' unplayed tiles, including the teammate and the opponents, and the first team to reach the agreed-upon total wins the game. In case of a tie, where no tiles are playable, the player or team with the fewest combined points from their unplayed tiles wins the round. If there is still a tie, the round is considered a draw, and the next round is played to break the tie [2].

For the implementation of the DominoAI game an online article was consulted that had a simple design. It provided a starting point for the development, but was also simple enough for us to create our own base code without it being entirely the same [3]. Additionally, an adjusted version of the rules were created. The revised version was adapted to accommodate local Puerto Rican rules, simplicity, and performance.

The revised version of the rules were the following:

1. First, the tiles are shuffled and each player is given 7 tiles.
2. The first play is determined by the player with the highest double tile.
3. The players match tiles to the open ends that have the same number on them, but if they can't play they can either draw an extra tile at random or pass their turn.
4. The game ends when a player runs out of tiles. If the game gets stuck the player with the least amount of points wins. Points refer to the total sum of the numbers in tiles in the player's hand.

These rules apply equally to both two and four player games, whether in free for all or team mode.

The performance measure for this exercise was simply described as the amount of steps taken to achieve this goal. The following formula was used to describe this behaviour:

$$Performance = \frac{Amount\ of\ wins\ obtained\ by\ the\ AI}{Number\ of\ games\ player} \times 100$$

III. Performance Approach and Results:

The developed Dominoes game had several modes and configurations to create different test environments for an AI. These modes would feature both 2 player and 4 player matchups as well as support for cooperative play. To allow diversity in team-based matchups, a feature was created to let human players pick their specific partner. The AI algorithm utilized Monte Carlo simulation, a technique that generates random samples from a probability distribution to approximate the behavior of a system or process. This is done by simulating numerous possible scenarios and picking the most optimal one [1]. The code implementation for this algorithm can be found in Figure 1. It is important to recognize that Monte Carlo does not guarantee the victory as Dominoes is a chance based strategy game.

The implementation includes the following gamemodes:

2 Player Mode:

- 1 AI vs 1 Human
- 2 AI (spectate)

4 Player Mode (All with team 2v2 support):

- 1 AI vs 3 Humans
- 2 AI vs 2 Humans
- 3 AI vs 1 Human
- 4 AI (spectate)

To measure the performance of the AI in different game modes and set ups, each game was executed three times. This allowed us to have multiple decisive samples of the performance without having extensive tests and still have a fair level of robustness. The results of which can be seen in Table 1. By using these results for the different modes available, it can be seen that AI fared better the more players there were playing, regardless if there were human or other AI agents. The amount of simulations done by the Monte Carlo simulation was 30 simulations to keep the program light weight and keep the computation time low enough so that it would not affect the flow of the game, especially when there are 3 AI agents all making 30 simulations each. In simulations without teams, the AI beat the human players 2 out of 3 times in the 4-player configuration. In 2-player games, the AI struggled to outperform the human due to the

increased complexity introduced by unpredictable tile draws, adding a level of randomness it couldn't anticipate. In 4-player configurations, the AI typically outperforms human players in both free-for-all games and team setups. It often held an advantage due to its ability to simulate up to 30 potential games with its current hand to determine the best possible move.

The results indicate that the AI's performance is primarily influenced by two key factors: the initial tile distribution and the number of simulations executed per turn. First, the starting tiles and the tiles drawn during the game, play a critical role. While Dominoes involves strategic decision-making, the options available to any player are inherently limited by the tiles they are dealt. This constraint is particularly important in four-player modes, where the inability to draw additional tiles limits the AI to relying solely on its initial hand. In such cases, the AI relies heavily on simulations to evaluate potential moves and estimate the most favorable outcomes possible.

The second factor affecting performance is the number simulations conducted during each turn. As mentioned previously, the AI was configured to perform 30 simulations per move, allowing it to explore a diverse range of scenarios and select the most advantageous option in the long run. As shown in Table 2 and Figure 2, increasing the number of simulations conducted during each turn significantly enhanced the AI's decision-making abilities. A higher number of simulations not only enhanced the AI's capacity to identify winning strategies, but also provided enough foresight to secure a stalemate or force a tie. Notably, when configured to run 10,000 simulations the AI became virtually unbeatable, with the best possible outcome for the human player being a tie.

To test the AI agent in its designed environment, a small tournament was established between the group members and the agent with both teams and free for all. During the tournament the agent struggled in the team section of 2 players and 2 AI as the attacking players often did not do the most optimal move as predicted by the agent making it struggle to compete and have to recalculate its moves. The same happened in the 4 player free for all as the non optimal nature of the human player moves left the AI unable to compute its most optimal strategy. However, in the case where the AI teamed up with a human partner the results were unexpected as it managed to win all 3 of the games. Adding the human unpredictability factor to the game in its strategy made its decision making more stable and complete. These results can be observed in Table 3.

The Monte Carlo simulation was chosen for its fair robustness and ability to perform effectively in different configurations. The amount of simulations per turn allowed us to increase or decrease the level of difficulty for the human players without having to restructure the entire code or adding too much complexity. Other adversarial search techniques were considered, but each of them came with disadvantages that outweighed their advantages. Stochastic games would have been the fastest to implement but it mainly considers randomness and doesn't make

for a realistic or formidable opponent [4]. Partially observable games are good for theoretical fit, but they are impractical for multiplayer games like Dominoes without severe approximations [5]. Average Over Clairvoyance assumes perfect knowledge which introduces bias towards unrealistic strategies [6]. Deep Neural Networks, also known as DNN, require large amounts of training data which might not be available for games like Dominoes. It also demands significant computational power which would make the game incompatible for some hardware [7]. Considering these variables, Monte Carlo simulations offered the best balance of adaptability and performance, making it the most practical and effective choice for our Dominoes AI implementation.

A. Figures and Tables

```python
def monte_carlo_ai_move(self, simulations=30):  1 usage  JanPerez35
    """
    Evaluate possible moves via Monte Carlo playouts and return best one.
    Currently, it runs 30 simulations per move.

    Args:
        simulations (int): Number of random playouts per move.

    Returns:
        Optional[Tuple[int, int]]: Best tile to play or None to pass.
    """
    hand = self.game.players[1]
    valid = self.game.get_valid_moves(hand)
    if not valid:
        return None

    scores = {}
    for m in valid:
        wins = 0
        for _ in range(simulations):
            sim = copy.deepcopy(self.game)
            try:
                sim.play_tile( player: 1, m)
            except ValueError:
                continue
            if self.simulate_random_playout(sim) == 1:
                wins += 1
        scores[m] = wins / simulations
    return max(scores, key=scores.get)
```

Figure 1. Monte Carlo Algorithm in Python

Game mode no teams	Performance out of 3 games for AI	Game mode in teams (2v2)	Performance out of 3 games for AI.
1 player VS 1AI	33.33%	N/A	N/A
2 AI (spectate)	66.67.%	N/A	N/A
3 players VS 1 AI	66.67%	3 players VS 1 AI	66.67%
2 players VS 2 AI	66.67%	2 players VS 2 AI	33.33%
1 player VS 3 AI	100.00%	1 player VS 3 AI	66.67%
4 AI (spectate)	100.00%	4 AI (spectate)	100.00%

Table 1. AI Performance Results in team-mode utilizing 2 and 4 Player games

Simulations for Monte Carlo (1 AI vs 1 Player Gamemode)	Performance out of 3 games for AI
30	33.33%
100	33.33%
500	66.67%
1000	66.67%
5000	66.67%
10000	100%

Table 2. AI performance utilizing different Monte Carlo Simulations

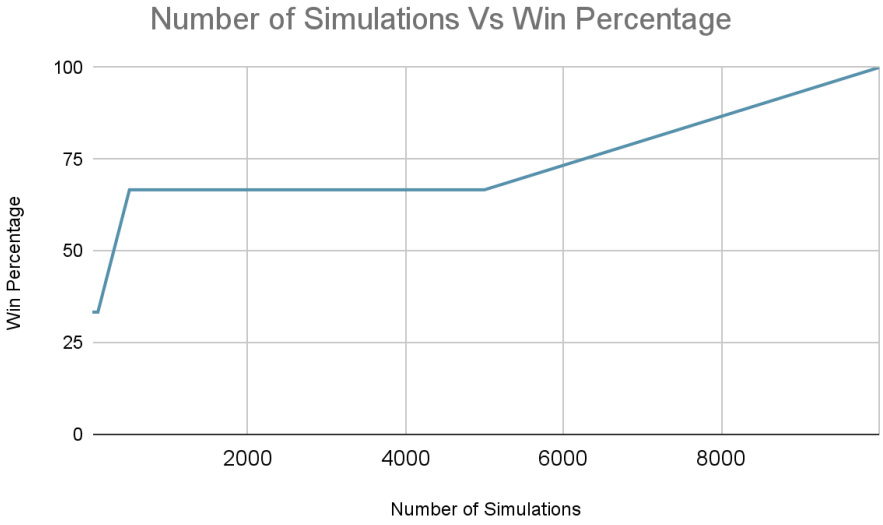

Figure 2. Number of Simulations vs Win Performance from Table

Game mode	Performance out of 3 games for AI
2 players VS 2 AI in teams	33.33%
4 player free for all	33.33%
4 player in teams	100.00%

Table 3. Tournament simulation with 3 Players and 1 AI

IV. Conclusion:

The objective of this programming assignment was to study adversarial search utilizing AI agents to solve or win games. The most prominent use of adversarial search in AI agents is within competitive scenarios. There were different methods to develop the adversarial search for the group's Domino game; them being Stochastic games, Partially Observable games , Monte Carlo simulation, Averaging over Clairvoyance, and Deep Neural Networks. For this interpretation the Monte Carlo simulation was chosen over the other options due to the other models requiring larger learning data sets, greater power limitations that affected the execution of the code, severe approximation, and introduction of bias towards unrealistic strategies due to estimations. To study the performance, a conglomerate of 3 games were tested for all the different modes in which the win ratio of the AI agent was measured and compared to the human players. Upon observing Tables 1 through 3 the results demonstrated that the AI struggles at predicting the tiles when multiple human players are inserted into the game. Its win rate

improves when it is transitioned to 4 player games without human intervention. The win rate rose to 66.67% of the games won by the AI agent for both 3 human players and 1 AI agent. This result shows that the AI is still able to calculate a probable win over the human players in both the individual games as well as in teams.

The AI model is still more capable than the human players, a regular human can't make 30 predictions in the span of one Domino turn. This means that under proper conditions such as the AI starting first or having a favorable starting hand, the win rate percentage increases greatly. By simulating the entire game, it can find the best move that leads to its victory utilizing inferences on the tiles the other players have and only placing the best tile that maximizes its chances of winning.

Even with its aforementioned strategy the AI agent is still capable of losing as can be seen in the test 2 human players vs 2 AI on team mode. Where the AI team only won 1 game out of 3, but the rest of the tests show that it has a clear advantage over human players especially when the simulation size is increased. The simulation was capped at 30 simulations per AI agent to limit the computational strain of the agent and for the smooth continuation of the game. When this cap was increased the computational abilities of the agent increased proportionally at the cost of computer runtime, making the agent's probability to win much higher, but the gameplay much slower. Since the functionality of the game was our main concern, we decided to keep the number of simulations low even if the AI's ability to win was lowered.

References:

[1] S. Sharma, “What is Monte Carlo Simulation,” *geeksforgeeks.org*, Aug. 26 ,2024. [Online]. Available:https://www.geeksforgeeks.org/what-is-monte-carlo-simulation/ . [Accessed Apr.19 , 2025].

[2] Domino Rules , “What is Monte Carlo Simulation,” *dominorules.com*, 2016. [Online]. Available: The Basics . [Accessed Apr. 19, 2025].

[3] A. Golovin, “Dominoes game with simple AI in Python,” *medium.com*, Jun 7, 2022. [Online]. Available: Dominoes game with simple AI in Python | by Aleksandr Golovin | Medium .[Accessed Apr. 19, 2025].

[4] M. Gupta, “Stochastic Games in Artificial Intelligence,” *geeksforgeeks.org*, Sep 11, 2024, [Online]. Available: Stochastic Games in Artificial Intelligence | GeeksforGeeks .[Accessed Apr. 19, 2025].

[5] L. Kaelbling, M. Littman, A. Cassandra, “Planning and acting in partially observable stochastic domains” *sciencedirect.com*, Jan. 17, 1998, [Online]. Available: Planning and acting in partially observable stochastic domains - ScienceDirect .[Accessed Apr. 19, 2025].

[6] G.Piliouras, R.Sim, S.Skoulakis, “Beyond Time-Average Convergence: Near-Optimal Uncoupled Online Learning via Clairvoyant Multiplicative Weights Update,” *arxiv.org* , Jun 7, 2022, [Online]. Available:[2111.14737] Beyond Time-Average Convergence: Near-Optimal Uncoupled Online Learning via Clairvoyant Multiplicative Weights Update.[Accessed Apr. 20, 2025].

[7] D. Silver et. all, “ Mastering the game of Go with deep neural networks and tree search,” *nature.com*, Jan 27, 2016. [Online]. Available: Mastering the game of Go with deep neural networks and tree search | Nature .[Accessed Apr. 20, 2025].

Code Repository:

https://github.com/JanPerez35/DominosAI.git