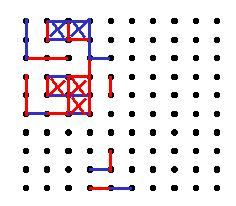
## horizontal line



Dots and Boxes

**Description:** There is a grid of dots, and each player gets to place a line between two adjacent dots, if a player places a line such that a box is created they get a point and the next turn is theirs again.



Sahil Kumar- 27149

Syed Muhammad Hussain-26992

# Problem Statement

Creating an AI that can play dots and boxes.

**Dots and Boxes Game Overview:**

**Dots and Boxes** is a classic pencil-and-paper game that is simple to learn and can involve deep strategic play. It is played by two players on a grid of dots.

**Game Objective:**

The main objective of Dots and Boxes is to complete more boxes than your opponent by drawing lines between adjacent dots.

**Setup**

1. **Grid**: Game starts with a grid of dots. The size of the grid can vary (e.g., 3x3, 4x4, etc.).
2. **Players**: Two players take turns. One player can be designated as Player 1, and the other as Player 2.

**Gameplay**

1. **Drawing Lines**: Players take turns drawing a single horizontal or vertical line between two adjacent dots.
2. **Completing a Box**: If a player draws a line that completes the fourth side of a 1x1 box, that player colors in the box (or marks it with their initial) and takes another turn.
3. **Extra Turns**: Completing a box allows the player to take another turn. This continues until the player is unable to complete another box on their turn.
4. **End of Game**: The game ends when no more lines can be drawn (i.e., all possible boxes are completed).

**Scoring**

* The player with the most completed boxes at the end of the game wins.

# Objectives

The goal of this project is to develop an AI capable of playing Dots and Boxes effectively. To achieve this, the AI must meet several specific objectives:

* **Correctly Evaluate the Current State**:

The AI should be able to assess the current configuration of the game board accurately, determining the number of boxes each player has completed and identifying potential moves.

* **Find the Next Best Possible Move Given a State**:

The AI should be able to analyze the game board and determine the optimal move to make. This involves strategic thinking to maximize the number of boxes completed by the AI while minimizing the opportunities for the opponent.

* **Calculate the Next Move in a Short Amount of Time**:

Efficiency is crucial. The AI should be capable of making decisions quickly to ensure a smooth and seamless gaming experience.

# Planning it out

* Our general approach was to break the problem into sub parts, develop a sub part, test it extensively and then to build the next sub part on top of that.
* First create the game board
* Create a function to input human move.
* Then check if moves are being registered on the board.
* Next check if points are being registered.
* Implement the evaluation function.
* Is the Evaluation function working properly?
* Implement AI using minimax and evaluation function
* Implement alpha-beta pruning
* Introduce difficulty levels.
* Implement the Graphical User design for the game.

# Methodology

To develop a proficient AI for playing Dots and Boxes, we implemented a structured approach covering board initialization, move registration, state evaluation, and AI decision-making. Below are the detailed steps and functions used in the implementation:

**1. Board Initialization**

**2D Array Representation**:

* We used a 2D array to represent the game board. Each cell in the array represents a dot, and the connections between the dots are stored to represent the lines drawn by the players.
* **initialize\_board()**: This function sets up the initial empty board. The structure of the array was carefully designed to keep track of the lines and the ownership of completed boxes.

**2. Move Registration**

**Human Player Move Input**:

* **make\_a\_move()**: This function allows the human player to input their.
* Initially it worked by allowing the human player to specify the x and y coordinates of the starting dot and the direction of the line ('r' for right, 'd' for down, etc.).
* After the implementation of the GUI, it read human player’s move from it.

**Box Completion Check**:

* **check\_point2()**: After each move, this function checks the four possible directions (up, down, left, right) around the starting dot to see if a box has been completed. If all four sides of a potential box are drawn, it marks the box with the current player's identifier.
* If a box is completed, the current player gets an additional move, which is crucial in maintaining the correct game flow.

**3. State Evaluation**

**Score Calculation**:

* **check\_score()**: This evaluation function calculates the current score by iterating over the board and counting the boxes completed by each player. The score is returned as the difference between the AI's score (score2) and the human player's score (score1). A positive value indicates the AI is winning, while a negative value indicates the human player is winning.

**4. AI Implementation**

**Minimax Algorithm with Alpha-Beta Pruning**:

* The core decision-making algorithm is based on the minimax method with alpha-beta pruning to optimize the search process. The implementation ensures the AI efficiently explores possible moves and evaluates their outcomes.
* The primary challenge addressed was the difference in turn mechanics between Dots and Boxes and other games like tic-tac-toe. Specifically, in Dots and Boxes, a player can continue to move if they complete a box. This was handled by incorporating the **check\_point2** function within the minimax algorithm to determine if a point was scored and whether the same player should move again.

**Difficulty Levels**:

* **Easy Mode**:
  + **check\_easy\_score()**: This evaluation function uses **check\_score()** but includes an element of randomness. The AI is programmed to make suboptimal moves 70% of the time, simulating a lower skill level.
* **Medium Mode**:
  + **check\_medium\_score()**: This function directly uses the **check\_score()** function, providing a balanced level of difficulty without additional randomness.
* **Hard Mode**:
  + **check\_hard\_score()**: This advanced evaluation function builds on **check\_score()** by also considering positions with three or two sides already drawn around a box. This provides a more precise assessment of potential future moves, allowing the AI to make more strategic decisions.

**Search Depth Variation**:

* The search depth for the minimax algorithm is varied based on the difficulty level. Easy mode has the shallowest search depth, medium mode has a moderate depth, and hard mode has the deepest search depth, increasing the complexity and accuracy of the AI's decisions.

# Implementation Steps

1. **State Representation:**

Design a data structure to represent the grid and track the current state, including which lines have been drawn and which boxes have been completed.

1. **Move Generation**:

Implement a function to generate all possible moves from the current state.

1. **State Evaluation**:

Develop an evaluation function to score each potential move.

1. **Minimax with Alpha-Beta Pruning**:

Implement the minimax algorithm with alpha-beta pruning to efficiently explore the game tree and find the best move.

1. **Testing and Tuning**:

Test the AI against various opponents and scenarios to fine-tune the evaluation function optimal performance.

# Results

We achieved all of our main objectives:-

1. Made an Ai that can play against a human player.
2. Implemented different difficulty levels successfully. Winning in hard mode is quite a challenge.
3. The Ai is Fast. In easy and medium mode it barely takes a second to make a move. In hard mode it takes max 2 to 3 seconds to make a move towards the end of the game(when depth > 6).

# Major Problems Encountered

During the development and implementation of the AI for playing Dots and Boxes, several we encountered major problems. Each of these issues required careful analysis and troubleshooting to ensure the AI functioned correctly and efficiently. Below is an evaluation of the key problems we faced and the solutions we came up with:

**1. Depth Search Limitation**

Problem:

Whenever there were only a few moves left, the program would return an error. This issue occurred because the depth of the search was set to 3, but when the number of remaining moves was less than 3, the program could not search at the specified depth, causing it to crash.

Fix:

We implemented a condition to handle this scenario:

if moves\_left < depth:

depth = moves\_left

This adjustment ensured that the search depth dynamically adapted to the number of moves left in the game, preventing the program from attempting to search beyond the available moves.

**2. AI Stopping Good Moves After Gaining a Lead**

**Problem**: After the AI scored a significant number of points, it would stop making optimal moves. The underlying reason was that the AI, having already secured more than half of the available points, would no longer prioritize making good moves, as it perceived the game to be already won.

**Fix**: To address this, we modified the AI's evaluation strategy to ignore past points and focus solely on potential future gains. This ensured the AI continued to strive for the best possible moves regardless of the current score:

**3. Board Size and Computational Limits**

**Problem**: The AI struggled to keep up with larger board sizes. While the AI could make very good moves on a large board, making the optimal move required searching at a very high depth (more than 10), which significantly slowed down the AI and diminished the enjoyment of the game.

**Fix**: To balance performance and playability, we decided to limit the maximum board size to 6x6. This compromise allowed the AI to perform efficiently while still providing a challenging and enjoyable experience:

# Evaluation

The development of an AI for playing Dots and Boxes presented us with several challenges, which we effectively addressed through a combination of strategic adjustments and algorithmic improvements. This evaluation section provides a comprehensive review of the project's successes, the major issues encountered, and the solutions implemented to ensure the AI's performance met the desired criteria.

**Successes**

1. **Effective State Representation**:
   * The use of a 2D array to represent the game board allowed for efficient tracking of moves and the status of each box.
2. **Accurate Move Validation and Scoring**:
   * The functions **check\_point2()** and **check\_score()** ensured that moves were correctly validated and that the game state was accurately evaluated. This was crucial for both fair play and the proper functioning of the AI's decision-making process.
3. **Implementation of Minimax with Alpha-Beta Pruning**:
   * The minimax algorithm, enhanced with alpha-beta pruning, allowed the AI to make strategic decisions efficiently. This combination ensured that the AI could evaluate numerous potential moves quickly, maintaining a high level of play without excessive computational delays.
4. **Difficulty Levels**:
   * By implementing different difficulty levels through varying search depths and evaluation functions, the AI was made accessible to players of varying skill levels. This added a layer of adaptability and challenge, enhancing the overall gaming experience.

**Overall Performance**

Our AI developed for Dots and Boxes successfully meets the project's objectives by correctly evaluating the current game state, identifying optimal moves, and making decisions in a timely manner. The adaptive strategies implemented ensure that the AI remains competitive throughout the game, providing a challenging opponent for players. The inclusion of difficulty levels allows the AI to cater to a wide range of player skills, enhancing the game's accessibility and enjoyment.

# Reflection

The development of the Dots and Boxes AI provided us valuable insights into the complexities of game strategy and algorithmic efficiency. Reflecting on the project, there are several areas for future enhancement and experimentation.

**Future Enhancements**

1. **Scaling AI Performance for Larger Boards**:
   * **Current Limitation**: The AI's performance decreases as the board size increases beyond 6x6 due to the need for deeper searches, which significantly slows down decision-making.
   * **Future Goal**: Implement an advanced algorithm capable of making optimal moves even on larger boards. This could involve refining the evaluation function to be more efficient and scalable.
     + **Potential Approach**: Develop a more sophisticated evaluation function that incorporates advanced heuristics and pattern recognition. This would allow the AI to assess complex board states more accurately without the need for exhaustive searches.
     + **Machine Learning Integration**: Explore the integration of machine learning techniques to train the AI on larger datasets, enabling it to learn optimal strategies from extensive gameplay data.
2. **Exploring Alternative Algorithms**:
   * **Minimax Limitations**: While the minimax algorithm with alpha-beta pruning has been effective, it may not be the most efficient or robust solution for all scenarios.
   * **Future Goal**: Experiment with other algorithms to evaluate their performance in Dots and Boxes. This includes both classical algorithms and modern approaches.
     + **Monte Carlo Tree Search (MCTS)**: Investigate the use of MCTS, which combines the precision of tree search algorithms with the robustness of statistical sampling. MCTS has shown promise in complex decision-making environments and could enhance the AI's performance.
     + **Reinforcement Learning (RL)**: Implement RL algorithms that allow the AI to learn from interactions with the game environment. Techniques like Q-learning or deep reinforcement learning could enable the AI to develop strategies dynamically through trial and error.
3. **Algorithm Competitions**:
   * **Current Focus**: The project has primarily focused on optimizing a single AI using the minimax algorithm.
   * **Future Goal**: Facilitate competitions between different algorithms to assess their relative strengths and weaknesses.
     + **Implementation**: Develop a framework that allows different AI algorithms to compete against each other in a series of games. This would provide empirical data on their performance and strategic differences.
     + **Analysis**: Analyze the outcomes of these competitions to identify which algorithms excel under specific conditions and why. This analysis could inform further improvements and hybrid approaches that combine the best features of multiple algorithms.
4. **Multi-Player Game Development**:
   * **Current Limitation**: The current game setup supports only two players, limiting the scope of gameplay.
   * **Future Goal**: Develop a version of the game that supports more than two players, including the AI as one of the players. This expansion would increase the game's complexity and appeal.
     + **Three-Player Mode**: Implement a game mode where three players can participate, with one being the AI. This mode would require rethinking the game dynamics and strategies, as players would need to consider the actions of two opponents instead of one.
     + **User Interface and Rules Adaptation**: Modify the user interface to accommodate additional players and adapt the game rules to ensure fair and balanced gameplay. This includes redefining the turn order, scoring system, and move validation for a multi-player setup.

# Conclusion

Overall, our project demonstrates a robust approach to AI development for Dots and Boxes. By addressing critical issues through innovative solutions, the AI not only performs effectively but also enriches the gaming experience for users. The balance between strategic depth and computational efficiency achieved in this project serves as a strong foundation for future enhancements and applications in AI-driven game development.