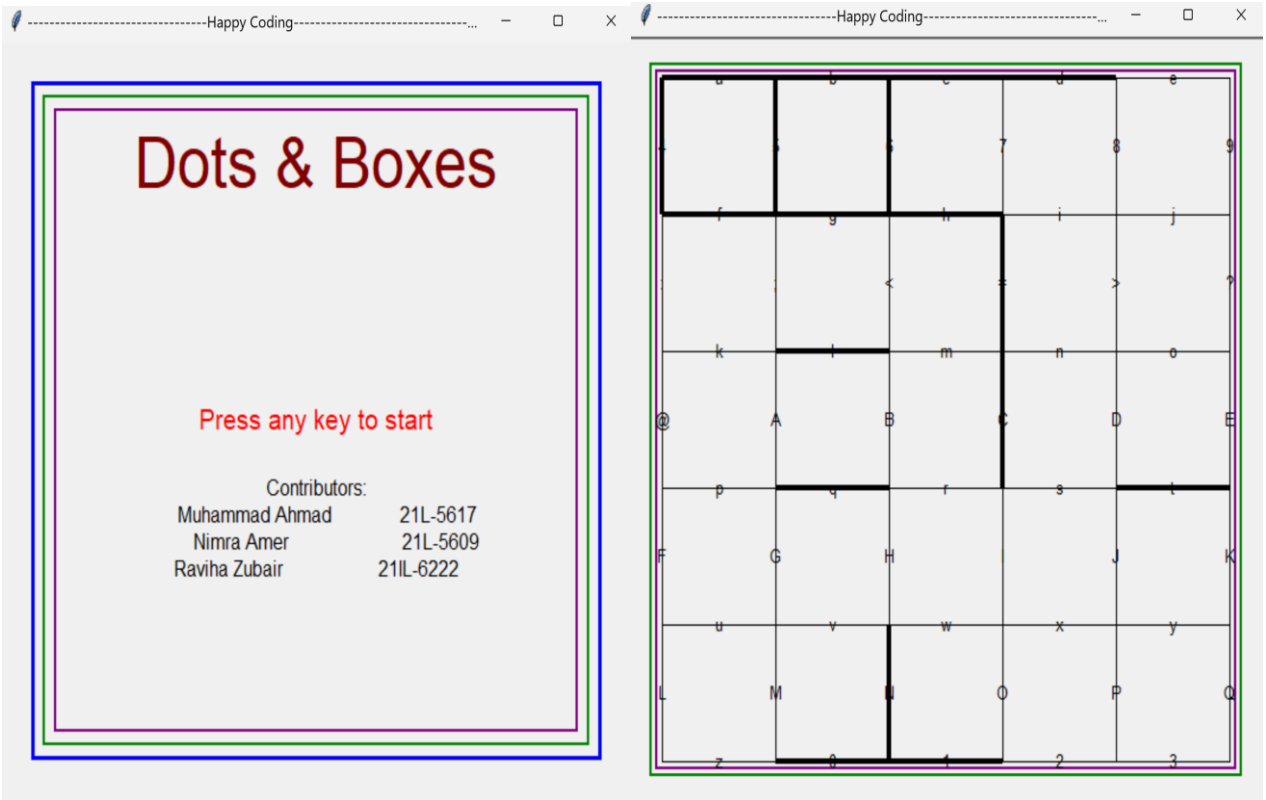
Dots and Boxes

Group members:

Muhammad Ahmad	21L-5617
Nimra Amer	21L-5609
Raveeha Zubair	21L-6222



Abstract:

This report presents the development and implementation of a computerized version of the game 'Dots and Boxes'. The game involves two players taking turns to connect dots on a grid, aiming to create squares and earn points. The implementation utilizes the MINIMAX algorithm to create playing agents for both human and AI players. The report discusses the game's rules, the structure of the implemented code, including the MINIMAX algorithm, and provides insights into the game's graphical user interface (GUI) using Python's Tkinter library. Additionally, it highlights the evaluation process for determining completed squares and summarizes the contributions of the project's developers. Overall, this report offers a comprehensive overview of the game's development, implementation, and key features.

Introduction:

The game is a classic two-player game played on a grid of dots. Players take turns connecting adjacent dots, with the goal of creating squares. Each completed square earns the player a point, and the player who completes the most squares wins the game. In this report, we present the development and implementation of a computerized version of Dots and Boxes featuring both human and AI players.

The implementation of the game incorporates various components, including the game board representation, player interaction, scoring system, and an AI agent capable of playing against human players. One of the key aspects of the implementation is the utilization of the MINIMAX algorithm, a decision-making algorithm widely used in two-player zero-sum games, to create an efficient AI player capable of making strategic moves.

This report provides an in-depth analysis of the game's rules, the structure of the implemented code, and the GUI developed using Python's Tkinter library. Additionally, it delves into the evaluation process for determining completed squares.

The Game Dots and Boxes:

The computerized game brings the classic pen-and-paper game to digital platforms, offering players the convenience of enjoying the game on their devices. Featuring an AI opponent powered by the MINIMAX algorithm, players can test their skills

against intelligent computer adversaries. With a user-friendly interface and automatic scoring, the game provides an engaging and interactive experience. Players can choose the grid size and difficulty level, tailor-making the game to their preferences. The computerized version offers strategic challenges and entertainment for players of all skill levels, preserving the fun and excitement of the traditional game in a modern digital format.

Rules and Characteristics of Dots and Boxes:

In the game implemented in the provided code, players must adhere to specific rules during gameplay. Firstly, each player must draw one line during their turn, connecting two adjacent dots on the grid. Diagonal lines and lines connecting non-adjacent dots are not permitted. Completing a box by drawing the fourth line between four dots earns the player a point and grants them another turn to draw an additional line. Once all possible lines have been drawn, the game concludes, and the player with the most completed boxes wins.

Dots and Boxes distinguishes itself from other games by its deterministic nature. Unlike games involving chance elements like dice or cards, every action in this game contributes to the outcome, even the seemingly random initial lines drawn by players. Additionally, the game is categorized as a combinatorial game, played sequentially with each player taking turns. It also qualifies as a perfect information game, meaning there are no hidden elements, and both players have full knowledge of the game state at all times. These characteristics add strategic depth and challenge to the gameplay experience, requiring players to think ahead and anticipate their opponent's moves.

Implementation:

The game is implemented using the MINIMAX algorithm for AI decision-making. It involves a graphical interface using tkinter and pygame for the game display and sound effects. The game logic is structured around generating and evaluating potential moves to maximize points by completing squares on the game board.

Code Structure:

The code is organized into separate modules, each responsible for a specific aspect of the game. This modular structure promotes code readability, maintainability, and scalability, making it easier to

understand and extend the functionality of the game.

Firstly, the **game_board.py** file defines the GameBoard class responsible for generating the game board grid and evaluating completed squares. This class handles the game state and provides methods for updating the board and evaluating the completion of squares.

Secondly, the **minimax.py** file contains the implementation of the MINIMAX algorithm for the game AI. It defines the minimax function responsible for determining the best move for the AI player. Additionally, it includes helper functions for generating possible moves and evaluating the game state.

Thirdly, the **main.py** file orchestrates the game flow. It sets up the graphical interface using tkinter and pygame, handles user input, and manages the interaction between the players and the game board. This file also includes functions for drawing the game grid, handling game over conditions, and alternating between human and AI turns.

In addition, the game features immersive sound effects using **pygame's** sound module. These effects, triggered during pivotal moments like making moves or completing squares, enhance the overall user experience by adding auditory feedback, making the gameplay more engaging and enjoyable.

Algorithm Used:

The implemented Minimax algorithm, residing in the **minimax.py** file, serves as the core decision-making mechanism for the AI player in the game. Operating on the principle of recursive exploration, Minimax traverses potential future game states up to a specified depth, systematically analyzing the consequences of each possible move. This depth parameter regulates the extent of exploration, balancing computational resources with decision accuracy.

Within the Minimax framework, evaluation functions play a pivotal role in assigning scores to individual game states. These functions serve as heuristics, providing a quantitative measure of the desirability of a particular game state from the perspective of the AI player. The evaluation function in the code assesses game states based on the number of completed squares achieved by the AI player. This metric indicates strategic advantage, with higher scores assigned to states where more squares are completed. By prioritizing moves leading to square completion, the function guides the

AI player towards maximizing its chances of winning.

Furthermore, the Minimax algorithm operates on the principle of zero-sum games, where the utility gained by one player is directly offset by the utility lost by the opponent. This competitive dynamic ensures that the AI player's decisions are not solely focused on maximizing its own gains but also on minimizing the opponent's potential advantages. Consequently, the AI player aims to maintain a strategic balance between offensive and defensive maneuvers, strategically navigating the game board to secure victory while thwarting the opponent's progress. Through the meticulous application of Minimax, the AI player endeavors to exhibit intelligent and adaptive gameplay, challenging human opponents and providing a stimulating gaming experience.

Evaluation Methods:

In the implementation, two evaluation methods are employed to assess the progress of players in completing squares within the game board. The primary method, encapsulated within the Evaluate Squares function in the **game_board.py** file, is responsible for determining the number of squares completed by a player following a move. This function carefully examines the game state after each move, identifying any newly completed squares and allocating points accordingly. By leveraging this method, the game logic effectively tracks players' advancements and rewards them based on their achievements, facilitating strategic decision-making.

Additionally, an alternative evaluation approach is presented through the Square Evaluator class in the **square_evaluation.py** file. Although not actively utilized within the game's core logic, this class offers a direct method for evaluating squares by inspecting the current state of the game board. While not integral to gameplay, this class serves as a valuable tool for debugging and verification purposes, providing an alternate perspective on square completion that may aid in refining the game's mechanics or troubleshooting potential issues. Overall, these evaluation methods significantly contribute to the robustness and versatility of the game implementation, enhancing both its functionality and maintainability. Their integration ensures a comprehensive approach to assessing player moves and game states, further enriching the gaming experience.

User Interface:

The user interface of the game is constructed using Tkinter, a Python library renowned for its simplicity and versatility in creating graphical user interfaces. Within this interface, a straightforward canvas-based grid is presented to users, serving as the primary arena for gameplay interactions. Through intuitive mouse clicks on the grid cells, players can execute their moves, marking their territory and strategically advancing their positions on the board.

Moreover, to facilitate single-player matches against an AI opponent, the interface seamlessly integrates the Minimax algorithm, automating the decision-making process for the AI player. This allows for a dynamic and engaging gaming experience, where users can challenge themselves against an intelligent virtual adversary. The implementation of the Minimax algorithm ensures that the AI's moves are calculated with precision, adding depth and complexity to the gameplay.

Audio features are seamlessly integrated into the game using Pygame's sound module, providing feedback on player actions and enhancing the overall immersion. From subtle cues for player interactions to rewarding sounds for significant events like completing squares, the audio elements enrich the gaming experience, engaging players and reinforcing their progress throughout the game.

To enhance the visual appeal and user experience, the interface incorporates subtle yet effective visual cues. Bold lines are utilized to demarcate completed sides of squares, providing players with immediate feedback on the progress of the game. These visual indicators not only serve a functional purpose in highlighting important game elements but also contribute to the overall aesthetic appeal of the interface, making the gaming experience more immersive and enjoyable.

Conclusion:

In conclusion, the Dots and Boxes game implementation delivers an immersive and enjoyable gameplay experience suitable for both human and AI opponents. Leveraging the Minimax algorithm ensures that players face a formidable challenge, enhancing the strategic depth of each match. With its blend of strategic gameplay and intuitive design, Make Squares promises hours of entertainment for players of all skill levels.