

Project Report
Artificial Intelligence
[CSE3705]

Othello

By

V.Sai Sumanth
210257

D.Veera Harsha Vardhan Reddy
210258



Department of Computer Science and Engineering
School of Engineering and Technology
BML Munjal University
November 2023

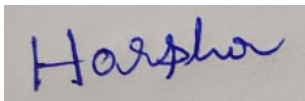
Declaration by the Candidates

We hereby declare that the project entitled "*Othello*" has been carried out to fulfill the partial requirements for completion of the course **Artificial Intelligence** offered in the 5th Semester of the Bachelor of Technology (B.Tech) program in the Department of Computer Science and Engineering during AY-2023-24 (odd semester). This experimental work has been carried out by us and submitted to the course instructor *Dr. Soharab Hossain Shaikh*. Due acknowledgments have been made in the text of the project to all other materials used. This project has been prepared in full compliance with the requirements and constraints of the prescribed curriculum.

V Sai Sumanth

V. Sumanth

D Veera Harsha Vardhan Reddy

A rectangular box containing a handwritten signature in blue ink that reads "Harsha".

Place: BML Munjal University

Date: 17 November, 2023

Contents

	Page No.
1. Introduction & Problem Statement	4-6
2. Methodology	7-12
2.1 Table Driven	
2.2 Intelligent Agent	
3. Implementation - Technology Stack	13
4. Conclusions	14
References	15
Appendix:	16

Introduction & Problem Statement

Introduction:

Othello, also known as Reversi, is a classic strategy board game played between two players on an 8x8 unchecked board. The game utilizes two-coloured pieces – black and white – with each player assigned a colour. The primary objective is to have the majority of one's colour discs on the board at the end of the game, achieved by flipping the opponent's discs to your colour through strategic placement of your own discs.

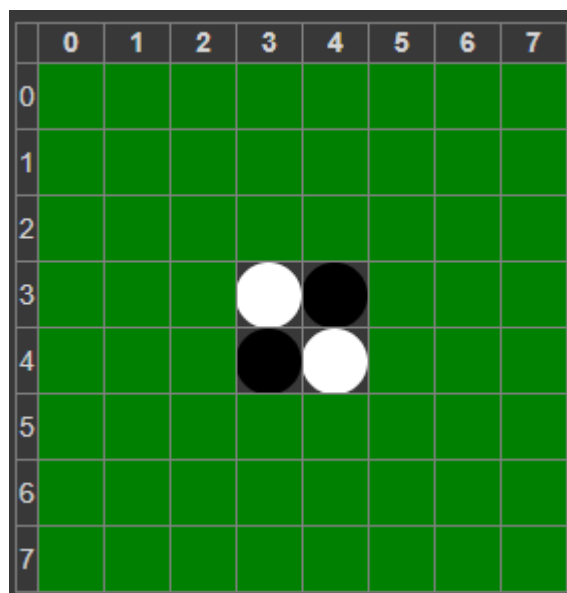


Fig 1: Initial position of board of Othello

Gameplay Mechanics:

Starting Position: The game begins with four discs placed in the centre of the board in a square pattern – two black and two white, with opposing colours diagonally adjacent.

Player Turns: Players alternate turns, placing one disc on the board per turn, with the requirement that the placed disc must flank one or multiple of the opponent's discs, forming a straight line (horizontal, vertical, or diagonal).

Flipping Discs: Once a disc is placed, all the opponent's discs that are flanked by the newly placed disc and another disc of the same colour, are flipped to match the colour of the placed

disc.

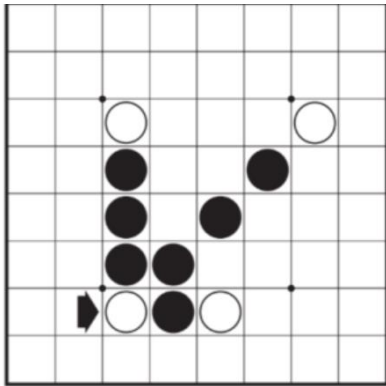


Figure 2: Disc placed

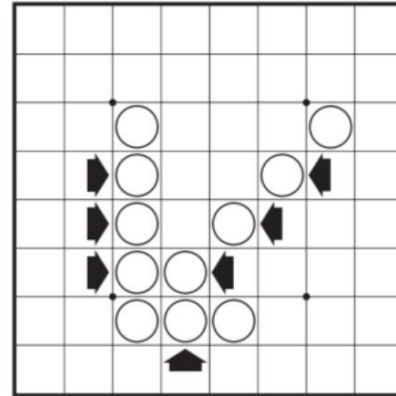


Figure 3: These discs flipped

End of Game: The game concludes when neither player can make a valid move, typically when the board is full, or no more flanking moves are possible.

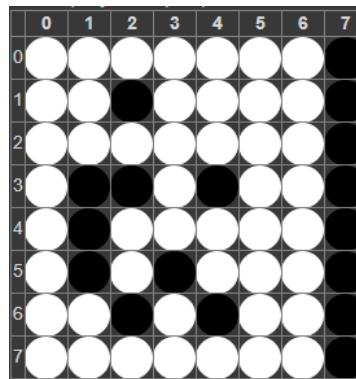


Fig: End game board representation

Strategy and Complexity:

Othello is deceptively simple in its rules but offers deep strategic complexity. Players must balance offensive and defensive strategies, anticipating the opponent's moves and understanding the long-term implications of each move. Positional play, control of key squares, and timing of flips are crucial aspects of high-level Othello strategy.

Problem Statement

Background

Othello is a strategy-based board game characterized by its simple rules yet complex gameplay dynamics. The game's outcome heavily depends on the players' ability to think ahead, strategize, and adapt to rapidly changing situations on the board. While human players can learn and improve their skills over time through practice and experience, replicating this level of strategic thinking and adaptability in an artificial intelligence (AI) agent presents a unique challenge.

Core Problem

The primary problem in developing an AI agent for Othello is to create a system that not only understands the fundamental rules and objectives of the game but also possesses the ability to make strategic decisions akin to an experienced human player. This involves several key challenges:

1. Understanding Game Mechanics: The AI must be programmed to comprehend the basic mechanics of Othello, including legal move determination, game board state evaluation, and understanding the rules for flipping opponent discs.

2. Strategic Decision Making: Unlike games of chance, Othello requires strategic foresight and planning. The AI agent must be capable of making decisions that consider not only the immediate effects of a move but also its long-term implications on the game.

3. Adaptability and Learning: A significant aspect of playing Othello effectively is the ability to adapt strategies based on the opponent's actions. The AI agent needs to be designed to recognize and respond to various strategies employed by the opponent.

Solution Approach

The solution to these challenges involves creating an AI agent that can effectively analyse the game board, anticipate future moves, and make decisions based on a comprehensive understanding of Othello's strategic nuances. The AI should be able to evaluate the consequences of each move and devise strategies that optimize its chances of winning, all while responding adaptively to the human opponent's gameplay.

2. Methodology

2.1. Table-Driven Approach

Overview

The table-driven approach in your Othello AI project involves defining specific percept and actions. Percepts are essentially the observations or inputs the AI agent receives from the game environment, while actions are the responses or operations the agent performs based on these percepts. This approach is advantageous for structuring and simplifying the decision-making process of the AI agent.

Utility of Table-Driven Approach

1. Structured Decision Making:

By mapping percepts to specific actions, the AI agent can make decisions in a more structured and organized manner.

2. Scalability and Modification:

It's easier to scale or modify the AI's behaviour by updating the tables rather than changing the core logic.

3. Clear Separation of Logic:

This approach provides a clear separation between the game's state (percepts) and the actions taken, facilitating easier debugging and understanding of the AI's behaviour.

Percepts and Actions

Percept	Description	Action	Description
valid_move	Checks if a move is valid at the specified position for the player.	make_move	Executes a move if it is valid.
opponent	Identifies the opponent player based on the current player.	flip_discs	Flips the discs in response to the opponent's move.
found_opponent	Checks if an opponent's disc is found in a line from a given position.	flip_discs	Flips the discs between two discs of the current player.
valid_coordinates	Ensures the coordinates are within the board limits.	make_move	Executes a move if the coordinates are valid.
player_disc	Checks if a disc at a position belongs to the player.	print_board	Displays the current state of the game board, highlighting the player's discs.
no_valid_moves	Determines if there are no valid moves available for the player.	pass_turn	Passes the turn to the opponent when no moves are available.
game_end	Checks if the game has reached an end state.	declare_winner	Declares the winner based on the board state at game end.
opponent_move	Observes the move made by the opponent.	analyze_move	Analyzes the opponent's move to adjust strategy.
board_full	Checks if the game board is fully occupied.	declare_winner	Declares the winner when the board is full.
capture_opportunity	Identifies an opportunity to capture multiple opponent discs.	make_strategic_move	Makes a move that maximizes the capture of opponent discs

2.2. Pattern Recognition in Othello AI Agent Using Predicate Logic

Pattern Recognition

Implementation of Othello employs elements of intelligent agent design, specifically incorporating knowledge-based reasoning and predicate logic. Here's why:

Knowledge-Based Reasoning:

Pattern Recognition and Recording:

The AI recognizes patterns in the game state and records them to make informed decisions. It maintains a history of moves and outcomes, inferring strategies based on past experiences.

Decision Making:

The AI evaluates possible moves based on past patterns and game history to make strategic decisions, selecting moves that have higher win rates or align with recognized patterns.

Predicate Logic:

Controlled Position Predicate: The function `controlled_position()` checks if a position on the board is controlled by a player based on the rules of the game. It uses predicates and logical checks to determine this condition.

Logical Rules for Moves:

The function `infer_move()` uses logical rules and conditions based on game state and previous moves to decide the best move. It applies predicate logic to assess the board and make decisions.

Overall:

The use of history tracking, pattern recognition, and logical inference to make decisions aligns with the characteristics of intelligent agents. Predicate logic is utilized to define rules and conditions for move selection and pattern recognition. Hence, this implementation showcases elements of intelligent agent behaviour in a knowledge-based and predicate logic-driven context within the game of Othello.

Overview

Pattern recognition in the Othello AI agent involves using predicate logic to identify and respond to patterns on the game board. This aspect of the methodology focuses on how the AI interprets the board's state and makes strategic decisions based on recognizable patterns.

Part 1: Predicate Logic in Pattern Recognition

1. Controlled Position (Predicate Logic):

The `controlled_position` function uses predicate logic to determine if a specific board position is under the control of a player. It does this by examining all directions from the given position and checking if a line of opponent's discs is followed by the player's disc. This logic is crucial for understanding board dominance and influence.

2. Corner and Edge Moves (Strategic Positioning):

Corner Move: The `corner_move` predicate checks if a move is in one of the four corners of the board. Corners are strategically significant in Othello due to their stability and control over adjacent lines.

Edge Move: The `edge_move` predicate evaluates whether a move is along the edge of the board. Edge positions are important for board control and often influence the game's outcome.

3. Opponent's Last Move (Reactive Strategy):

The `opponent_last_move` function serves as a reactive strategy. By identifying the opponent's last move, the AI can adjust its strategy accordingly. This predicate is critical in the AI's ability to adapt to the opponent's gameplay.

Part 2: Implementation of Predicate Logic

1. Infer Move Based on Patterns: The 'infer_move' function embodies the AI's pattern recognition capability. It evaluates the current board state, the player's position, and the opponent's recent moves. The function decides the AI's next move based on the analysis of these patterns, combining strategic positioning with reactive strategy.

2. Recording Patterns and Their Outcomes: The 'record_pattern' function records observed board patterns along with their outcomes. This historical data forms the basis for the AI's learning mechanism, where it can refer to past patterns and their effectiveness.

3. AI Move Decision Based on Pattern Analysis: 'get_ai_move' utilizes the recorded patterns and current game state to determine the AI's next move. This decision-making process is guided by the analysis of the opponent's last move, the strategic importance of corners and edges, and the historical success rate of similar patterns.

Part 3: Implementation of min-max

For more and better intelligent agent adding min max approach give better results

Evaluation Function (evaluate)

- This function calculates a heuristic evaluation of the current game state.
- It considers:
- **Disc Difference:** The difference in the number of discs between the AI and the opponent. A positive value favours the AI, negative favours the opponent.
- **Corners:** Occupying corner positions is highly advantageous in Othello, so the function assigns high scores if the corners are occupied by the AI.
- **Edges:** Occupying edge positions is also beneficial, though less so than corners. This function assigns scores for occupying edges.

Minimax Algorithm with Alpha-Beta Pruning (minimax)

- **minimax** is a recursive algorithm that explores the game tree by considering all possible moves up to a certain depth.
- It performs a depth-limited search, evaluating each possible move's outcome based on the evaluation function.

- It alternates between maximizing and minimizing the potential outcome for the AI and the opponent, respectively.
- The **alpha** and **beta** values are used to prune branches of the tree that are not promising, reducing the number of nodes evaluated.
- It utilizes **has_valid_moves** and **is_valid_move** functions to check for valid moves and if the move is valid for a given player.

AI Move Decision (get_ai_move)

- This function utilizes the minimax algorithm to decide the best move for the AI given the current board state and the player.
- It calls **minimax** with a depth of 4 (meaning it looks ahead four moves) to decide the best move.

The AI iterates through all valid moves, simulates each move on a copy of the board, evaluates the resultant position using the **evaluate** function, and selects the move that leads to the most favourable outcome (based on the heuristics and the minimax algorithm's search).

This approach doesn't consider all possible moves to the end of the game (which is often infeasible due to the game's complexity) but instead tries to find the best move within a limited lookahead, using the evaluation function to estimate the desirability of a given board position.

Implementation - Technology Stack for Othello AI Agent

Overview

The technology stack for implementing the Othello AI agent comprises various tools and technologies, each playing a crucial role in the development and operation of the AI. This stack is selected to optimize the performance, flexibility, and user interaction of the AI agent.

Core Technologies

1. Programming Language: Python is the primary programming language used for the AI agent. It's chosen for its simplicity, readability, and extensive support for data structures and algorithms, which are essential for AI development. Python's extensive libraries and frameworks, particularly those for AI and machine learning, provide strong support for implementing complex algorithms and AI strategies.

2. Game Board Visualization: HTML is used for visualizing the Othello game board within the **Jupyter** Notebook environment. HTML provides a flexible way to create an interactive and user-friendly interface. The Jupyter Notebook offers an interactive development environment, allowing for real-time testing and modification of the AI's code and strategies.

3. AI Decision-Making Algorithms: Custom algorithms implemented in Python drive the AI's decision-making process. These include pattern recognition logic, strategic evaluation, and move prediction. The algorithms are designed to process game states, evaluate possible moves, and make strategic decisions, mimicking a human player's thought process.

4. Data Structures and Algorithmic Libraries:

Python's built-in data structures like lists, dictionaries, and tuples are extensively used for storing game states, patterns, and move histories.

Libraries such as NumPy, Math are potentially used for efficient numerical computations, especially for handling multi-dimensional arrays representing the game board.

Conclusion

Insights into the Othello AI Agent Project

Our intelligent agent's effective implementation combines predicate logic and table-driven approaches in an efficient manner to provide strong pattern recognition. Our agent quickly makes decisions by efficiently analysing and interpreting game data through the use of precomputed tables. Predicate logic integration gives the agent the ability to identify and react to complex patterns in the game environment, allowing for a deeper understanding of strategic arrangements.

Moreover, the use of the Minimax method, strengthened by alpha-beta pruning, increases the agent's capacity for making decisions. The algorithm methodically assesses possible moves through a depth-limited search, thereby balancing the pursuit of maximising gains for the agent and minimising losses against the opponent. This strategic insight plays a crucial role in directing the agent towards the best possible gameplay.

Essentially, our intelligent agent demonstrates a well-balanced combination of table-driven techniques, predicate logic for pattern identification, and the effective Minimax algorithm's decision-making powers. By serving as a reliable guide, the heuristic evaluation function helps the agent maximise its strategic ability by navigating the complexities of the gaming environment.

References:

1. World Othello Federation. (n.d.). Official Rules of Othello. Retrieved from World Othello Federation: <https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english>
2. Know It All Ninja. (n.d.). Pattern Recognition, Generalisation & Abstraction. Retrieved from Know-It-All Ninja: <https://www.knowitallninja.com/lessons/pattern-recognition-generalisation-abstraction/>
3. Scaler Academy. (n.d.). Predicate Logic in AI. Retrieved from Scaler Academy: <https://www.scaler.com/topics/artificial-intelligence-tutorial/predicate-logic-in-ai/>

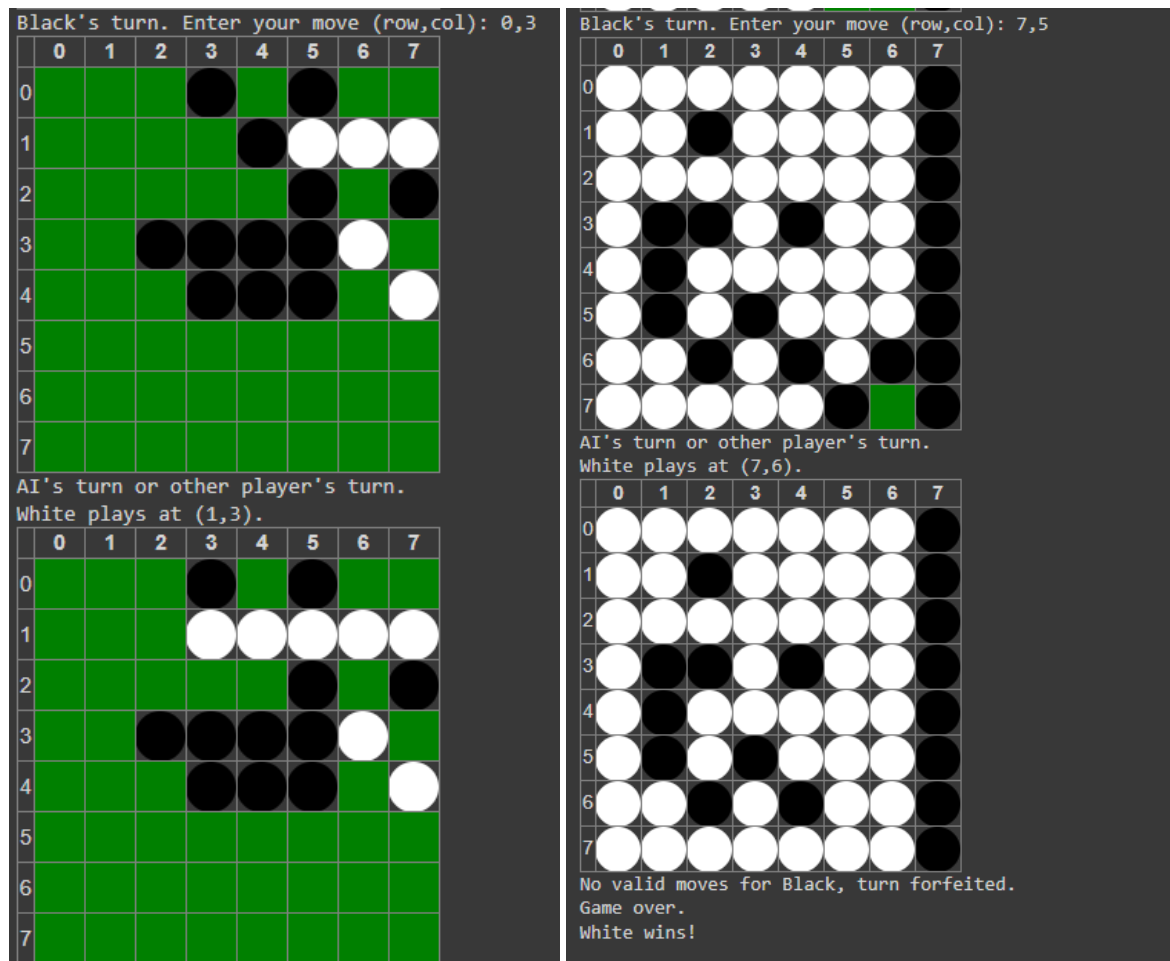
Appendix

Collab Notebook:

<https://colab.research.google.com/drive/1m-AS4AdfmnxBCFNYr6HUFah9gNRmguWa?usp=sharing>

Output:

As the Othello game is big to show at one go these are some and full video is given below.



Full game play video:

<https://clipchamp.com/watch/adIhtvDxEuy>