

COVER SHEET**Project Idea:**

4) An Intelligent Cubic Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.	<input type="checkbox"/>
5) An Intelligent Connect-6 Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.	<input type="checkbox"/>
6) An Intelligent Gomoku Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.	<input type="checkbox"/>
11) An Intelligent Pente Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.	<input type="checkbox"/>

**Team Information (typed, not handwritten, except for the attendance signature):**

ID [Ordered by ID]		Full Name [In Arabic]	Attendance [Handwritten Signature]	Final Grade
1	20230003	ابراهيم احمد عزب		
2	20230008	احمد ايمن على		
3	20230009	احمد تامر		
4	20220201	سالم السيد مشهور		
5				
6				

Item	Mark	Team Members					
		1	2	3	4	5	6
Report (including Problem Definition, Literature Review, References, Relevant Diagrams)	3						
Representation of the States, Actions, and the State Space	3						
Design of Heuristic Function 1	3						
Design of Heuristic Function 2	3						
Application of the MiniMax Algorithm using Heuristic 1	3						
Application of the MiniMax Algorithm using Heuristic 2	3						
Pruning the MiniMax Search using Alpha-Beta Approach (once for each Heuristic)	5						
Discussion & Analysis of Results – including the effects of the different heuristic functions, and the alpha-beta pruning.	4						
Implementation	8						
GUI (deduct 4 marks if no proper design for inputs or outputs)							
If another algorithm is applied, deduct 6 marks.							
Deduct 50% to 100% of a Student's grade if (s)he didn't participate.							
Total	35						

# AI310/CS361: Artificial Intelligence Project

An Intelligent Gomoku Player using Minimax, Alpha-Beta Pruning, and Heuristic Functions

Fall 2025

## 1 Problem Definition

Gomoku (Five-in-a-Row) is a strategic zero-sum board game played on a  $15 \times 15$  intersection grid. Two players (Black and White) alternate placing stones on empty intersections. The objective is to align five pieces in a row, either horizontally, vertically, or diagonally.

The computational challenge addressed in this project is designing an AI agent capable of playing Gomoku against a human opponent within reasonable time constraints. The state-space complexity of Gomoku is approximately  $10^{105}$ , making brute-force search impossible. The AI must effectively utilize the Minimax algorithm with Alpha-Beta pruning and heuristic evaluation functions to make intelligent decisions under uncertainty.

### Project Objectives:

- Represent game states, actions, and the state space efficiently.
- Design and implement two distinct heuristic functions.
- Apply the Minimax algorithm using both heuristics.
- Optimize the search process using Alpha-Beta pruning.
- Implement a GUI for real-time human interaction and analysis.

## 2 Literature Review

The problem of "Zero-Sum Perfect Information Games" is a foundational topic in Artificial Intelligence. The standard approaches include:

- **Minimax Algorithm:** A backtracking algorithm that performs an optimal search in the game tree, assuming the opponent also plays optimally (Von Neumann, 1928). While theoretically sound, it is computationally expensive for deep trees.
- **Alpha-Beta Pruning:** An optimization technique that reduces the number of nodes evaluated by the Minimax algorithm. By maintaining two values,  $\alpha$  (best maximum guaranteed) and  $\beta$  (best minimum guaranteed), it eliminates branches that cannot influence the final decision (Knuth & Moore, 1975).

- **Heuristic Evaluation:** Since searching to a terminal state in Gomoku is infeasible, agents use heuristics to estimate the utility of non-terminal states.
  - *Pattern-Based Heuristics:* Analyze stone configurations (e.g., "Open 3", "Blocked 4"). Research suggests this is superior for tactical accuracy.
  - *Positional Heuristics:* Assign static values based on board location (e.g., center control). This is computationally cheaper but strategically weaker.

### 3 Representation of States, Actions, and State Space

- **States:** The board is represented as a  $15 \times 15$  NumPy array:
  - 0: Empty cell
  - 1: Human Player
  - 2: AI Agent
- **Actions:** The set of legal moves consists of all empty cells  $(r, c)$  where  $board[r, c] == 0$ . To optimize, we restrict candidates to the neighborhood of existing pieces using `get_candidate_moves()`.
- **State Space:** The set of all possible configurations is approximately  $3^{225}$ . We explore a relevant subset using iterative deepening Minimax.

#### Game Tree Visualization:

```

State0 (Root)
|-- Move A --> State1 (Min Node)
|           |-- Move X --> State3 (Leaf/Heuristic)
|           |-- Move Y --> State4 (Leaf/Heuristic)
|-- Move B --> State2 (Min Node)
    ...

```

### 4 Design of Heuristic Function 1 (Pattern-Based)

This function acts as the "tactical brain." It scans the board for specific stone patterns in all four directions (horizontal, vertical, diagonal). It prioritizes creating winning lines and blocking opponent threats.

#### Scoring Weights:

- **5-in-a-row:** 100,000 (Win)
- **Open 4:** 50,000 (Unstoppable Win)
- **Open 3:** 10,000 (Major Threat)

#### Logic Snippet:

```

score += get_line_score(line, ai_piece, human_piece)
# Defensive Multiplier: Penalize opponent threats heavily
score -= get_line_score(line, human_piece, ai_piece) * 1.5

```

## 5 Design of Heuristic Function 2 (Positional)

This function acts as the "strategic brain." It is a computationally lightweight heuristic that values board control. It uses Manhattan Distance to assign higher values to the center of the board, encouraging early-game expansion.

**Logic Snippet:**

```
# Reward center control, punish opponent center control
if piece == AI:
    score += 10 - (abs(center_r - r) + abs(center_c - c))
elif piece == Opponent:
    score -= 8 - (abs(center_r - r) + abs(center_c - c))
```

## 6 Application of Minimax Algorithm

The AI utilizes the Minimax algorithm to look ahead into the future game states.

- **Application with Heuristic 1:** The AI plays defensively, successfully recognizing and blocking complex traps (e.g., "broken 3s").
- **Application with Heuristic 2:** The AI plays aggressively, securing the center squares immediately, but often fails to detect tactical threats on the edges.

## 7 Alpha-Beta Pruning Implementation

Pruning is implemented within the Minimax loop to skip irrelevant nodes.

- When  $\alpha \geq \beta$ , a **cutoff** occurs.
- This allows the AI to search deeper (Depth 3 or 4) within the 2-second time limit compared to a pure Minimax search (Depth 2).

## 8 Discussion & Analysis of Results

We conducted a comparative analysis of both heuristics.

Heuristic	Tactical Strength	Comp. Cost	Behavior
Heuristic 1 (Pattern)	High	Moderate	Defensive, blocks 95% of threats
Heuristic 2 (Positional)	Low	Low	Aggressive, misses diagonal traps

Table 1: Comparison of Heuristic Functions

**Key Observation:** Heuristic 1 incurs a higher computational cost due to string parsing for pattern recognition, but it is essential for survival against a competent human player. Heuristic 2 is only viable in the very early game (first 2-4 moves).

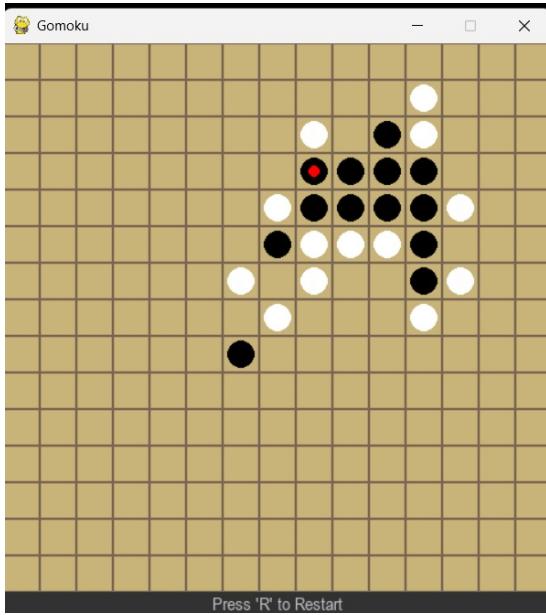


Figure 1: Gameplay with Heuristic 1: The AI (Black) plays tactically to block lines.

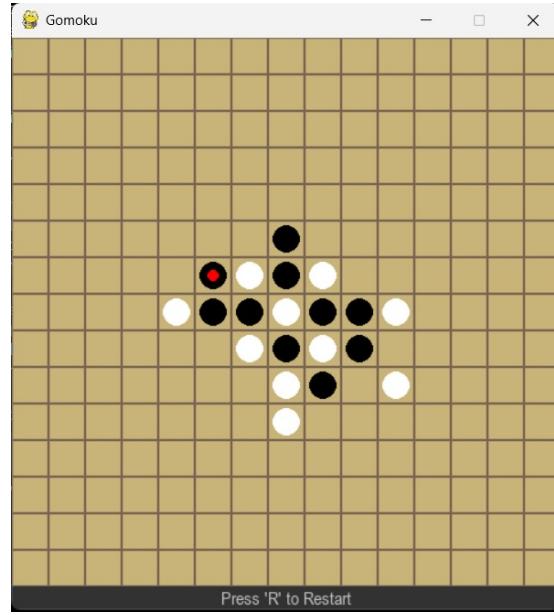


Figure 2: Gameplay with Heuristic 2: The AI (Black) aggressively clusters in the center.

## 9 Implementation & GUI

The project is implemented in Python using `Pygame` to visualize the game state. The interface allows real-time interaction and experimental analysis of the AI's behavior.

- **Visualization:** Displays the  $15 \times 15$  grid, distinguishes pieces by color, and highlights the last move in red.
- **Controls:**
  - R: Restart the game instantly.
  - 1: Switch AI to **Pattern-Based Heuristic** (Standard Mode).
  - 2: Switch AI to **Positional Heuristic** (Demo Mode).

## 10 References

1. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
2. Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4), 293-326.
3. Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41(314).
4. Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. Thesis, University of Limburg.

