Efficient Gomoku Agent Training via Negamax Algorithm with Alpha-Beta Pruning

Design and Specification Proposal
COMP702 – M.Sc. project (2024/25)

Submitted by
Junkang Hu
(201841383)

under the supervision of
Sven Schewe

Department of Computer Science
University of Liverpool

1 Statement of ethical compliance: A0

Data Category: A

Participant Category: 0

I confirm that I have read the ethical guidelines and will follow them during this project. This project does not involve the use of any personal or sensitive data, and no human participants will be involved at any stage.

2 Project description

This project describes a Gomoku agent that is optimised by Negamax algorithm and Alpha-Beta pruning. This agent will implement efficient strategy power that combines game tree search with heuristic evaluation functions. In addition, the agent will be trained through self-play without relying on human input, improving both algorithmic stability and generalization performance. Specifically, this project includes the following components:

- Building a Gomoku game environment that supports legal move validation and win detection;
- Implementing a decision-making algorithm based on Negamax algorithm and Alpha-Beta pruning to enhance search efficiency;
- Designing and tuning heuristic evaluation functions to assess board states effectively;
- Optimising the agent's policy through iterative self-play training;
- Evaluating the agent's performance by competing against random and baseline strategies.

This project aims to explore the effectiveness of classic search algorithms in Gomoku and improve playing performance under limited computational resources.

3 Aim and Requirements

3.1 The main aim

The main objective of this project is to develop an agent capable of playing Gomoku with good decision-making and strategic abilities. The agent will be built upon the Negamax algorithm, enhanced with Alpha-Beta pruning to achieve efficient decision-making. It will be trained through self-play to optimise its strategy and will be evaluated by competing against both random and baseline strategies to verify its practical performance.

3.2 Requirements

The requirements are:

- The system must support standard Gomoku rules, including legal move validation, win condition (five in a row), and board boundary checks.
- The AI agent must be based on the Negamax algorithm and optimized using Alpha-Beta pruning for efficient search.
- The search depth should support at least d=3 to ensure a certain level of strategic foresight.
- A heuristic evaluation function must be implemented to assess board positions in non-terminal states.

- The system should support self-play that is automated matches against both random and baseline strategies, with win rates recorded.
- A visualization interface should be included to display the board and AI decision process in real time.

4 Key literature and background reading

After rightly understanding the rule of Gomoku in [1], we will combine the Negamax algorithm [2] and Alpha-Beta pruning [3] methods to implement an agent and improve this agent through iterative self-play training [4].

4.1 Minimax tree search algorithm and Negamax algorithm

In the Minimax tree search algorithm [5], a game tree is constructed where each node represents a possible game state. The root node corresponds to the current state of the game, and the leaf nodes represent terminal game outcomes (win, lose, draw) or intermediate states evaluated when the maximum search depth is reached. These leaf nodes are assigned static values, which are calculated using a heuristic evaluation function.
The algorithm assumes a two-player, zero-sum game with alternating turns. These players are MIN and MAX, MIN aims to minimize the evaluation score and MAX aims to maximize the evaluation score, they exist alternately. For achieving these targets, the specific process is recursive backtracking. Starting from leaf nodes, at each MAX node, the highest value among its children is selected, while at each MIN node, the lowest value is selected until committing the result value to the root node. Finally, the best action for the current player is determined.

```
def minimax(state, depth, maximizingPlayer):
    if depth == 0 or game_over(state):
        return heuristic_evaluate(state)

    if maximizingPlayer:
        maxEval = -infinity
        for child in generate_moves(state):
            eval = minimax(child, depth - 1, False)
            maxEval = max(maxEval, eval)
        return maxEval
    else:
        minEval = +infinity
        for child in generate_moves(state):
            eval = minimax(child, depth - 1, True)
            minEval = min(minEval, eval)
        return minEval
```

Pseudocode 1: Minimax Pseudocode

The Negamax algorithm is a variant of the classical Minimax procedure, commonly used in two-player, zero-sum games such as board games. It leverages the mathematical property of zero-sum games:

$$\max(a, b) = \min(-a, -b)$$

This allows MIN to be transformed into (MAX - opponent score) so that all nodes become "maximize" without writing double recursive logic. As a result, the algorithm eliminates the need for separate logic for MAX and MIN players, simplifying the recursive implementation while preserving the same decision-making behavior.

```
def negamax(state, depth, color):
    if depth == 0 or game_over(state):
        return color * heuristic_evaluate(state)

    max_value = -infinity
    for child in generate_moves(state):
        value = -negamax(child, depth - 1, -color)
        max_value = max(max_value, value)

    return max_value
```

Pseudocode 2: Negamax Pseudocode

4.2 Heuristic evaluation function

Heuristic evaluation functions play an important role in guiding decision-making within adversarial search algorithms, especially when exhaustive enumeration of game states is computationally intractable, such as in some board games. In the context of two-player deterministic games with perfect information, such functions are designed to approximate the relative desirability of intermediate, non-terminal states. A well-designed heuristic can reduce the effective search depth required to identify optimal actions and improve convergence in algorithms such as Minimax or its variants.
The heuristic evaluation function provides a scalar utility estimate that reflects the likelihood of a favorable outcome for the evaluating agent. These estimates are typically informed by domain-specific features extracted from the game state. Nešić N. et al. [6] design a heuristic evaluation function framework specially focusing on board games to provide a fast and focused evaluation method for intermediate stages of heuristic function design. performance of search-based agents.

4.3 Alpha-Beta pruning

To find the best path, the Minimax tree search algorithm has to run through the whole game tree. The Alpha-Beta pruning technique simplifies the process and it will prun the tree nodes that have little probability of delivering a better move to avoid run towards them. Generally, when pruning results in bypassing an entire sub-branch of the game tree, it means that it can save a lot of time.
The Alpha-Beta pruning technique imports 2 parameters into the Minimax tree search algorithm, called Alpha and Beta. Largest value of maximizer at or above the stated level is Alpha, and the greatest value of minimizer at or above the specified level is Beta. As we known, the score rises positively for the maximizer and negatively for the minimizer, so that Alpha can be set as -INF, and Beta can be set as +INF, that means both players start with their lowest score.
Now we are going to talk about the condition of prunning. We will discard the subtree if one stage, that the highest score of the minimizer becomes less than the lowest score of the maximizer , is reached.

4.4 Self-play traning

Self-play is an open-ended multi-agent reinforcement learning centralized training paradigm [7]. It enables an agent to learn purely by simulating plays with a copy of itself, or fixed policies generated during training. In this process, the agent will maintain and update dynamically a benchmarking policy to remove dominated or redundant policies. Daniel Hernandez et al. [8] designed a a generalizing framework defined under formal notation to describe self-play algorithms

in multi-agent reinforcement learning. Although this project only involves a single intelligent agent, self-play is still an effective approach to improve the agent performance through iterative self-competition.

5 Development and Implementation Summary

The work plan can be summarised as follows.

- **Step 1: Environment Construction**
  Develop a Gomoku game environment, that specifically is an application software, that includes legal move validation, turn management, and automatic win condition detection. This environment will serve as the foundation for agent interaction and evaluation.

- **Step 2: Random and Baseline Strategies Opponents Development**
  Implement two types of opponent agents: a random move agent and a rule-based baseline agent. These opponents will be used both during self-play training (as optional adversaries) and in the final performance evaluation phase.

- **Step 3: Heuristic Function Design**
  Define and tune a heuristic evaluation function capable of estimating the value of non-terminal board states. The function will be used to guide the agent's search when a full-depth traversal is not feasible.

- **Step 4: Core Search Algorithm**
  Implement the Negamax tree search algorithm enhanced with Alpha-Beta pruning to reduce redundant computation and improve decision efficiency. This step ensures that the agent can explore deep game states with optimized pruning.

- **Step 5: Training Loop**
  In the beginning period of the project, the agent will compete against random and baseline strategies opponents. Then, when it has the basic ability to run a whole game, using a self-play framework in which the agent iteratively competes against copies of itself. This will allow the policy to improve over time without relying on external data or human supervision.

- **Step 6: Performance Evaluation**
  Benchmark the trained agent by playing against random and rule-based baseline agents. Analyze win rates and move quality to assess the effectiveness of the combined search and heuristic strategy.

The interconnection between all these steps is illustrated in the following figure.

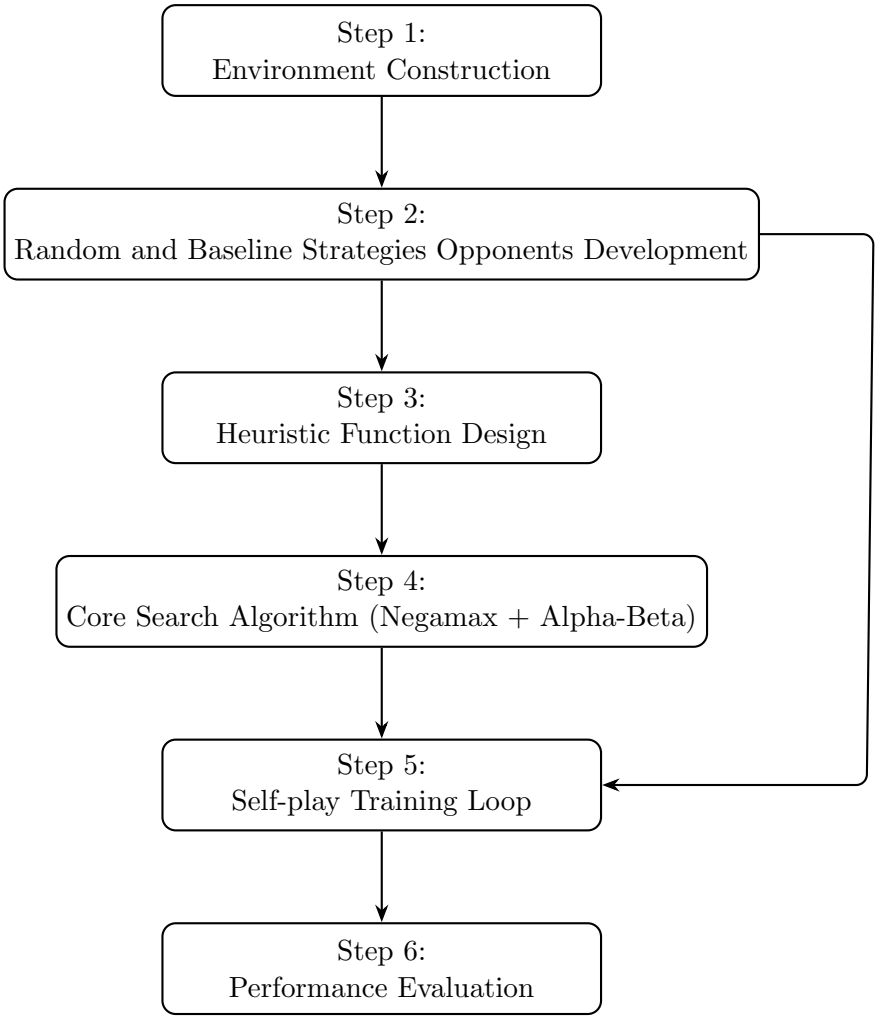


Figure 1: Development and Implementation Workflow

We will use some of the latest gadgets in our implementation such as:

- VSCode + Github Copilot and some other LLM agents.

6 Data source

This project does not require any external data or human participants. All training and evaluation data are generated internally within the custom Gomoku environment. The agent improves through self-play, interacting with itself or predefined rule-based opponents. This setup ensures full control over data generation and avoids ethical issues related to external or personal data sources.

7 Testing and evaluation

The agent will be tested and evaluated through the following procedures:

- **Functional Testing:** Ensure that the Gomoku environment operates correctly, including legal move validation, turn alternation, and automatic win/draw detection. Unit tests will be written for core game logic and agent decision-making modules.

- **Performance Evaluation Against Baselines:** The trained agent will be evaluated by playing multiple matches against two types of opponents: (1) a *random move agent* and (2) a *rule-based baseline agent* implementing basic offensive and defensive heuristics. Metrics such as win rate, average game length etc. will be recorded and analyzed.
- **Search and Evaluation Efficiency:** The effectiveness of the heuristic evaluation function and the search algorithm (Negamax with Alpha-Beta pruning) will be tested by measuring average search depth, computation time per move, and pruning ratio across sample games.
- **Robustness Through Self-Play:** The agent will also be tested in self-play settings to evaluate its learning stability and strategy diversity over iterations. This can help identify overfitting to specific opponents and ensure generalization.

All experiments will be reproducible, and evaluation results will be visualized to support analysis and comparison.

8 UI/UX mockup

The user interface for this Gomoku system will be minimalistic and functional, focusing on clear interaction and intuitive layout. The following elements will be featured:

- **Game Board Interface:** A 19x19 grid-based board, that refers to Chinese Go board, will be rendered where players and the AI can take turns placing black or white stones.
- **End Game Notification:** Once a win or draw condition is met, a popup or banner will indicate the outcome and offer options to restart or exit the game.
- **Basic Controls:** Buttons for starting a new game and toggling between self-play and demo modes.

The UI will prioritize ease of use and clarity, making the system accessible for both developers testing the agent and users observing the gameplay.

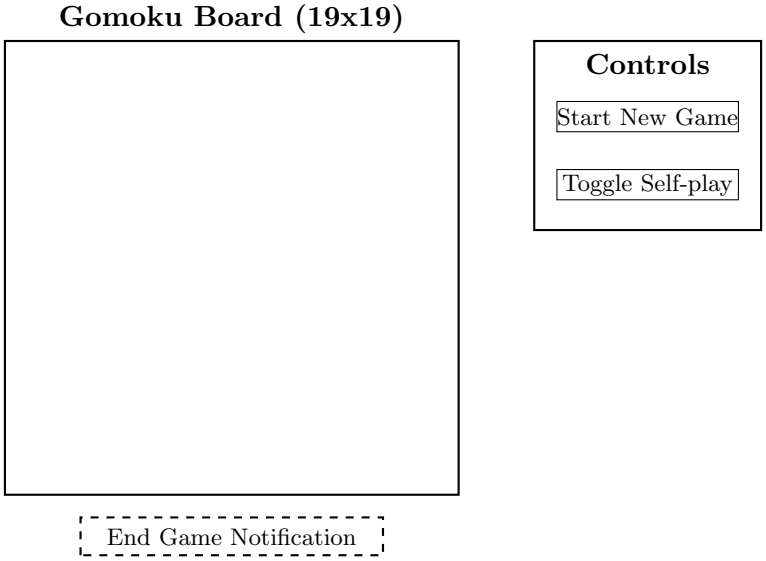


Figure 2: UI/UX Mockup of the Gomoku System

9 Project ethics and human participants

This project complies fully with ethical guidelines and does not involve the collection or processing of any personal or sensitive human data. The design and evaluation of the Gomoku agent are conducted entirely through programmatic simulation, without the need for external human participants. Specifically:

- No identifiable human data is used in any part of this project.
- All training and testing are performed using self-play between automated agents or predefined strategies (random and rule-based).
- There are no surveys, interviews, or interactive sessions involving human subjects.

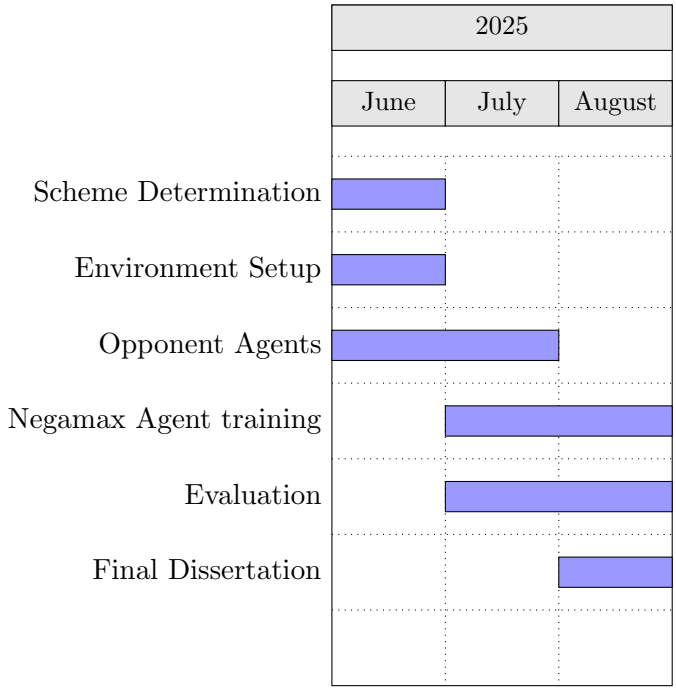
As such, the project falls under **Data Category A** and **Participant Category 0**.

10 BCS project criteria

1. Application of Practical and Analytical Skills:
   The project demonstrates the application of core algorithmic concepts such as heuristic search, adversarial decision-making, and tree-based evaluation, all of them are related to theoretical foundations acquired throughout the degree programme.
2. Innovation and/or Creativity:
   While based on classical AI algorithms, the project explores their adaptation to Gomoku with a custom-designed heuristic and a self-play training framework. The combination of strategic evaluation and simulation in a constrained, rule-based game context reflects creative integration of known techniques.
3. Synthesis and Evaluation of Solutions:
   The final system integrates multiple components, including a rule engine, search algorithm, and training loop, into a cohesive AI agent. The evaluation includes benchmarking against random and baseline agents, allowing for an basic assessment of the solution's effectiveness.
4. Real-World Relevance:
   Game AI research has broad relevance to strategic decision-making in fields such as robotics, autonomous systems, and online gaming. This project focus on examining how lightweight algorithms can deliver intelligent behavior under limited resources.
5. Self-management of Significant Work:
   The entire development lifecycle has been managed independently.
6. Critical Self-evaluation:
   The development process included iterative testing and refinement, with regular assessments of design choices, algorithm performance, and heuristic reliability. Limitations and trade-offs can not be avoided, and alternative strategies will be kept considered during the whole phases.

11 Project plan

The project will be carried out over a structured timeline that reflects the logical progression of work, allowing time for development, testing, and refinement.



12 Risks and contingency plans

There is the main technical risk and its possible contingency:

- Algorithm Performance Limitations: The Negamax with Alpha-Beta pruning may still not achieve sufficient search depth due to computational constraints, limiting the agent's strength.
  Contingency: Simplify or optimize the heuristic evaluation function to reduce computational load; consider depth-limited search and iterative deepening to balance depth and performance.

There are some possible project management risks and their corresponding possible solutions:

- Time Overruns: Underestimating time needed for algorithm tuning or self-play training.
  Contingency: Set weekly milestones to monitor progress and adjust workload.
- Insufficient Training Results: Self-play training may fail to improve the agent obviously.
  Contingency: Experiment with different heuristic weights and training parameters.

References

[1] Sakata, G., Ikawa, W. and Sloan, S. (1981) *Five-in-a-row. renju: For beginners to advanced players. Ishi Press International.*

[2] Knuth, D. and Moore, R. (1975) *An analysis of alpha-beta pruning, Artificial Intelligence* 6(4), pp. 293–326.

[3] Prof. Sumit S Shevtekar, Mugdha Malpe, and Mohammed Bhaila. (2022) *Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning. International Journal of Scientific Research in Computer Science, Engineering and Information Technology.* pp. 328–333.

[4] Samuel, A.L. (2000) *Some studies in machine learning using the game of checkers. IBM journal of research and development.* 44(1–2), pp. 206–226.

[5] Campbell, M.S. and Marsland, T.A. (1983) *A Comparison of Minimax Tree Search Algorithms. Artificial Intelligence* 20(4), pp. 347–367.

[6] Nešić, N. et al. (2016) *Heuristic Function Evaluation Framework. In Computers and Games. Cham: Springer International Publishing.* pp. 71–80.

[7] Balduzzi, D. et al. (2019) *Open-ended learning in symmetric zero-sum games.*

[8] Hernandez, D. et al. (2022) *A Comparison of Self-Play Algorithms Under a Generalized Framework. IEEE transactions on games* 14(2), pp. 1–1.