Spring 2025
Introduction to Artificial Intelligence

Homework 2: Connect Four

Due Date: 2025/04/04 (Friday) 17:00

1. Introduction

In this homework, you will develop and implement advanced adversarial search techniques for the game **Connect Four**. Your goal is to build an intelligent agent capable of evaluating board positions, planning ahead, and making optimal moves by employing classical search algorithms. Although you are provided with a skeleton codebase to verify your implementation, you must design and implement the core algorithms and a custom agent function from scratch.

Welcome to Connect Four

Connect Four is a two-player connection game that challenges players to strategically place their pieces on a grid in order to create a line of four consecutive discs. Here are the key details and rules of the game:

- **Game Board:**
  The game is played on a vertically suspended grid,with 6 rows and 7 columns.
- **Game Pieces:**
  Each player has a set of colored discs (red and yellow). Players take turns dropping one disc at a time into any of the columns.
- **Gameplay Mechanics:**
  - When a disc is dropped into a column, it occupies the lowest available space within that column due to gravity.
  - Players alternate turns, with each turn consisting of selecting a column to drop their disc.
- **Objective:**
  The primary objective is to be the first to form a horizontal, vertical, or diagonal line of four of your own discs. Achieving this creates a "Connect Four," and the game is immediately won by the player who accomplished it.
- **Game End Conditions:**
  - **Win:** The game ends as soon as a player successfully connects four discs in a row.
  - **Draw:** If the board is completely filled with discs and no player has achieved a connect four, the game is declared a draw.

2. Code Base Structure

This codebase is intended to run on a local machine and is not compatible with Google Colab due to its reliance on a graphical user interface (GUI). Ensure that you have Python 3 installed and are comfortable using the command line interface (CLI) to execute the programs.

The tested environment includes:

- **Python** 3.11.2
- **NumPy** 1.24.2
- **Pygame** 2.6.0

Other versions should also work, but make sure these packages are installed correctly.

Getting Started

- **Demo Game:**
  - Run `python connectFour.py` to start a demo game.
  - The demo launches a GUI where you can interact with the game using your mouse.
- **Reflex Agent:**
  - To try out the provided reflex agent, run the following command
  - Run `python connectFour.py -p ReflexAgent`
  - **Note:** The reflex agent serves as a baseline and tends to perform poorly even on simple cases.
- **Multiple Games & Fast Execution:**
  - You can play multiple games in one command by using the `-n` flag.
  - To run the games quickly without GUI, use `-q` flag to disable graphics.
  - Example : `python connectFour.py -p ReflexAgent -n [number_of_games] -q`
- **Change Agent:**
  - The default agent for Player 1 is Human-controlled/ ReflexAgent and the default agent for Player 2 is ReflexAgent.
  - You can change Player 1's Agent by `-p` flag, and change Player 2's Agent by `-e` flag.
  - There are 4 agents that you can use: `ReflexAgent, Minimax, AlphaBeta, StrongAgent.` We provide ReflexAgent for you, but the other agents should be implemented by yourself.
  - For example: `python connectFour.py -p ReflexAgent -e AlphaBeta`

Code base contains the following files:

- **connectFour.py:**
  This is the main entry point for running the game. It handles the GUI and overall game management. Use this file to launch demo games and test your agents.
- **agents.py:**
  This file is where you will implement your search algorithms (such as minimax, alpha-beta…) and design your evaluation function. All core logic related to decision-making should be implemented here. ***Do not rename or remove this file*** *as the autograder depends on it.*
- **game.py:**
  This file contains the Connect Four game engine, including functionalities for managing the board state, generating legal moves, and determining game outcomes (win, loss, or draw).

Additional Guidelines

- *You may create additional helper modules if needed; however, ensure they are included in your final submission.And do not use libraries in your assignment.*
- *All primary logic for search algorithms and the evaluation function must be implemented in **agents.py** as directed.*

3. Requirements

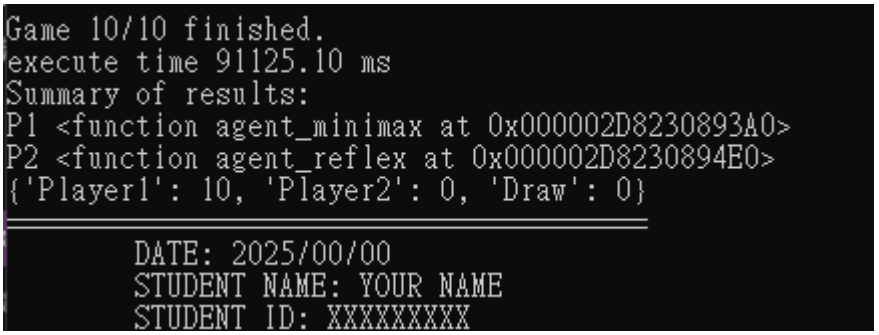
In this assignment, you will modify the provided **`agents.py`** file to implement adversarial search algorithms for **Connect Four**.

Important Rules:

- Modify the code within the markers in `agents.py`.
- **Do not import any external libraries**—use only the provided functions.
- Your search algorithms must support **arbitrary depth expansion**, meaning they should work for any given search depth.

Part 0: print_INFO

You must modify the `print_INFO()` function in `agents.py` to print the date and your information on the screen (see below example).
If you don't do this, you won't get any points.



Part 1: Minimax Search (10%)

Implement the **`minimax()`** function in `agents.py` to select the optimal move for **Connect Four** using Minimax search.

1. Recursive Minimax Search

- Recursively explore the game tree up to a given depth.
- Each level alternates between:

- **Maximizing player (agent's turn)** → Chooses the move with the **highest value**.
- **Minimizing player (opponent's turn)** → Chooses the move with the **lowest value**.

2. Handling Terminal States

- Stop searching when the game reaches a **win, loss, or draw** using `grid.terminate()`.
- If the maximum depth is reached, evaluate the board using `get_heuristic(grid)`.
- Use `game.drop_piece(grid, col)` to simulate a move.

3. Return Format

Your function must return a **tuple**:

- `tuple[0]`: The **evaluated board value** at the optimal move.
- `tuple[1]`: A **set of columns** that lead to this optimal value.

4. Testing

Run the following command to test against a simple agent:

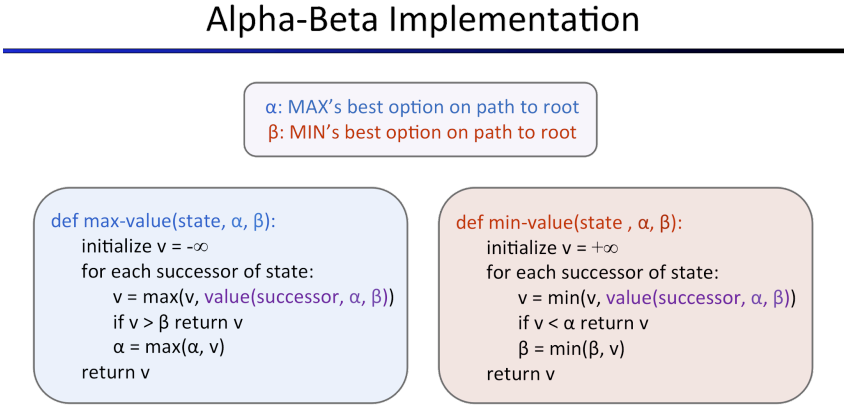
```
python connectFour.py -p Minimax -e ReflexAgent -q -n 100
```

Your **Minimax agent** should consistently **defeat ReflexAgent**.

```
Summary of results:
P1 <function agent_minimax at 0x105cc2fc0>
P2 <function agent_reflex at 0x105cc3100>
{'Player1': 10, 'Player2': 0, 'Draw': 0}
```

Part 2: Alpha-Beta Pruning (10%)

Implement the **`alphabeta()`** function in `agents.py` to optimize Minimax using Alpha-Beta Pruning.



1. Optimized Game Tree Expansion

- Identical to Minimax but **prunes unnecessary branches** using **α (alpha) and β (beta)**.

2. Alpha-Beta Pruning Logic

- **α (alpha):** Best maximizing move found so far.
- **β (beta):** Best minimizing move found so far.
- **Prune** branches where `beta ≤ alpha`, skipping unnecessary evaluations.

3. Return Format

Your function must return a **tuple** (same as Minimax):

- `tuple[0]`: The **evaluated board value** at the optimal move.
- `tuple[1]`: A **set of columns** that lead to this optimal value.

4. Handling Terminal Nodes

- Use `grid.terminate()` to detect win/loss/draw.
- Use `get_heuristic(grid)` to evaluate board states at depth limits.

5. Testing

Run the following command to compare efficiency:

```
python connectFour.py -p AlphaBeta -e ReflexAgent -q -n 100
```

Your **Alpha-Beta Pruning agent** should compute results **faster than Minimax** while maintaining identical move choices.

Part 3: Stronger Agent & Heuristic Function (10%)

Objective

Implement an advanced AI agent, **`agent_strong()`(Depth 4)**, that consistently outperforms the **Alpha-Beta Pruning agent (Depth 4)**, which moves first by default.

1. Improved Heuristic Function

You must implement **`get_strong_heuristics(grid)`** to enhance decision-making. This function should provide a **more refined** board evaluation than the default `get_heuristic()`.

Your heuristic should be optimized to counter the **Alpha-Beta (Depth 4) agent**, considering the following factors:

- **Winning Sequences & Forced Wins**:
  - Identify moves leading to a guaranteed win and prioritize them.
  - Recognize **imminent opponent victories** and **block them**.
- **Strategic Positioning**:
  - Favor **center control** for better future move flexibility.
  - Prioritize **moves that provide multiple winning options**.
  - Optimize positioning to **control key areas of the board**.
- **Depth-Aware Evaluation**:
  - Adjust heuristic evaluation based on **depth**, ensuring better **long-term planning**.

2. Implementation Requirements

- Your `agent_strong()` can either be the same as the Alpha-Beta Pruning agent (Depth 4) or an improved version using any techniques you choose. However, you must ensure that this agent also operates at Depth 4.
  - Only return **valid moves** from `grid.valid`.
  - Never **crash or raise exceptions**, regardless of the board state.
  - Use **Depth 4** for evaluation, ensuring fair comparisons.

3. Performance Benchmark

You must provide proof in your report that **`agent_strong()`** achieves:

- A **win rate above 50%** over **100 games** against the **Alpha-Beta (Depth 4) agent**.

4. Testing

Run the following command to evaluate performance:

```
python connectFour.py -p AlphaBeta -e StrongAgent -q -n 100
```

💡 **Note:**

- If your **Alpha-Beta agent (Part 2) is incorrect or does not use Depth 4**, you will receive only **partial credit** for this part.

Report (30%)

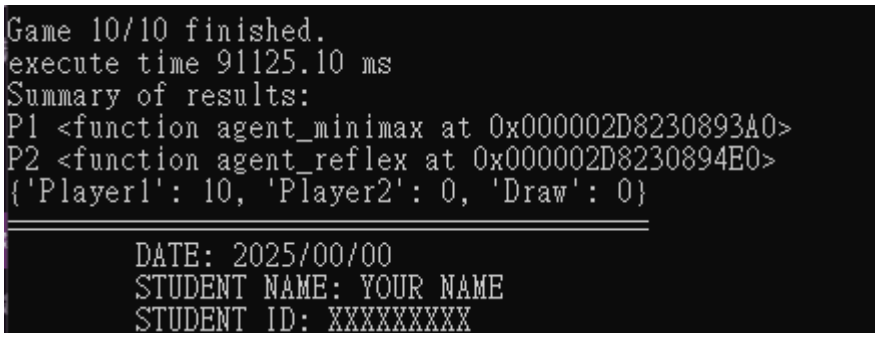
Your report should clearly present your implementation process, methodology, evaluation results, and insights gained from this homework. Below are the required sections and what you need to include in each.

1. Introduction (5%)

2. Implementation (15%)

2.1 Minimax Search (5%)

- **Explain the algorithm:**
  - How it recursively explores the game tree.
  - How it selects moves for the maximizing and minimizing players.
  - The role of `get_heuristic(board)` in evaluation.
- **Results & Evaluation:**
  - Run **Minimax vs. ReflexAgent** and show your screenshots of 100 game results (your minimax agent should be depth 4)
  - Your screenshots have to include your information, for example:



2.2 Alpha-Beta Pruning (5%)

- **Explain the optimization:**
  - How α-β pruning reduces unnecessary node expansions.
  - When and how pruning occurs in your implementation.
- **Results & Evaluation:**
  - Compare **execution time** of **Minimax vs. Alpha-Beta**.
  - **Plot: Execution Time vs. Search Depth** to show efficiency improvement.

2.3 Stronger AI Agent (`agent_strong()`) (5%)

- **Techniques Used:**
  - Explain how you developed `agent_strong()`, detailing the key strategies and optimizations implemented. Discuss the design choices made to improve decision-making, and heuristic evaluation.
- **Advanced Heuristic Function (`get_heuristic_strong()`):**
  - How it improves move evaluation over `get_heuristic(board)`.
  - How it **counters Alpha-Beta (Depth 4)** by prioritizing:
    - **Winning potential** (forced wins).
    - **Defensive play** (blocking imminent opponent wins).
    - **Board control** (favoring strategic positions like center dominance).
- **Results & Evaluation:**
  - **Show Win Rate of `agent_strong()` vs. Alpha-Beta (Depth 4)**.
  - Demonstrate that `agent_strong()` wins more than **50% of games over 100 games**.

3. Analysis & Discussion (5%)

- What were the difficulties in designing a strong heuristic?
- Did `agent_strong()` have any weaknesses (e.g., high computation time, failure in some cases)?
- How could `agent_strong()` be further enhanced?

4. Conclusion (5%)

5. References (if any)

Demo (40%)

We will host a demo session where you will be asked questions about your implementation and approach to this homework. Be prepared to explain your code, decision-making process, and the techniques you used.

QA Page

If you have any questions about this homework, please ask them on the following Notion page. We will answer them as soon as possible. Additionally, we encourage you can answer other students' questions if you can.

https://www.notion.so/HW2-QA-1bac9908e88680b68fabd187040461af?pvs=4

Submission

Due Date: 2025/04/04 (Fri.) 17:00

Please compress your agents.py and report (.pdf) into `STUDENTID_hw2.zip` and only submit `STUDENTID_hw2.zip` file

The file structure should look like:

```
📂{student_id}_hw2.zip
  ┣ 📜agents.py
  ┗ 📜report.pdf
```

Wrong submission format leads to -10 points.

Late Submission Policy

20% off per late day

Reference:

1. **Move Ordering in Search Algorithms**
   - **Chess Programming Wiki: Move Ordering**
     This resource provides a comprehensive explanation of move ordering techniques, detailing how prioritizing strong moves can significantly enhance the efficiency of Alpha-Beta pruning.
2. **Monte Carlo Tree Search (MCTS) and Its Enhancements**
   - **Fullstack Academy: Monte Carlo Tree Search Tutorial**
     This video tutorial explains the fundamentals of Monte Carlo Tree Search (MCTS), a widely used search algorithm in AI decision-making.