DS/CMPSC 442: Artificial Intelligence
Spring 2024
Project-2 Adversarial Search
(Due: March 26th 11:59 PM)

Question 1: Designing Wild Tic-Tac-Toe Agents

Figure.1 An example of playing tic-tac-toe

Project Description

We all know how to play tic-tac-toe. In this project, we will be implementing a variation of classic tic-tac-toe (let’s call that wild tic-tac-toe). In wild tic-tac-toe, the objective is to make three in a row without allowing your opponent to make three in a row on the next move. If a player makes three in a row, the opponent gets one more move to make three in a row themselves. If the opponent can do so, they win immediately, and the first player loses. Otherwise, the first player to make three in a row wins.

The game is played on the same 3x3 grid as the traditional version. Players take turns placing their respective symbols, commonly "X" and "O", on the board. Because a player can lose even after they make three in a row, the game requires an entirely different strategy, as players must avoid making three in a row if the opponent can do so as well on the next move. This game often leads to more complex strategies and counter-strategies, as players must constantly evaluate the risk and reward of each move. It introduces an intriguing twist to the familiar game and challenges players to think in a counterintuitive way. Luckily for me, I have my DS/CMPSC 442 students who can help me in designing an agent that plays this wild version of tic-tac-toe for me.

Deliverable	Explanation
File: tictactoe-ai.zip	A .zip file containing your modified python files. (You will receive a zip file containing a python implementation of a tic-tac-toe agent, which you will modify. Described below.)
File: chess-game-AI.zip	A .zip file containing updated python files for a chess AI program.
File: submission.pdf	It includes written answers for Questions 1.1, 2.1 and 2.2.

Table 1. Submission files for Project-2

Question 1.0 Setup Local Files (0 Points)

Instead of building a wild tic-tac-toe player from scratch, we will take an existing implementation that implements a minimax search based (classical) tic-tac-toe solver, and then we will modify it to solve the wild tic-tac-toe game.

As a first step, use these instructions to setup your local machine. You will need **Python 3.9 or higher**. Older versions of python will throw errors; these files are not backwards compatible.

1. Download the tictactoe-ai-master.zip file and Extract All.
2. Run *pip install pygame*
3. In order to run the game, navigate to the directory with main.py and write the command on terminal or cmd: *python main.py*

Note: You might run into some issues of missing packages, if you do run into them, you can install those missing packages first by running *pip install <package name>.*

To check whether you have successfully installed the dependencies, you can try playing a game of tic-tac-toe using the GUI application that pops up when run *python main.py*.

Question 1.1 (10 Points)

Update the engine.py file (in tictactoe-ai/tictactoe/engine.py) and board.py (in tictactoe-ai/tictactoe/board.py) file such that you can end up getting a tic-tac-toe AI solver that plays the wild version of tic-tac-toe.

Send us a .zip file containing your modified AI that plays the wild version of tic-tac-toe. We will play your AI agent to verify that your agent is playing wild tic-tac-toe in the correct way. We will play against your AI agent to see how well it plays.

Question 2: Designing Chess Agents

Figure.2 An example of playing chess (cited from www.chess.com)

Project Description

Chess is a board game for two players, called White and Black, each controlling an army of chess pieces, with the objective to checkmate the opponent's king. Chess is an abstract strategy game that involves no hidden information and no elements of chance. It is played on a chessboard with 64 squares arranged in an 8×8 grid. At the start, each player controls sixteen pieces: one king, one queen, two rooks, two bishops, two knights, and eight pawns. White moves first, followed by Black. The game is won by checkmating the opponent's king, i.e. threatening it with inescapable capture. There are also several ways a game can end in a draw.

If you unfamiliar with the rules of chess, please read this introductory article list to get a better understanding of the components of chess playing: https://www.chess.com/learn-how-to-play-chess

In this project, you are supposed to create your own chess playing AI agent. Instead of trying to build an AI agent from scratch, we are going to take an off-the-shelf chess playing agent, and then try to improve it by designing a better evaluation function. This chess-game-AI-master.zip file contains a Python based AI chess player that implements a depth-3 minimax search tree, and uses a naïve evaluation function to evaluate non-terminal states that are encountered at the end of 3 levels of the tree. Your ONLY job in this project question is to implement a new evaluation function that allows your AI agent to play better.

To implement my evaluation function, how many files do I need to modify?
Just one file. The evaluation function is defined inside the ***calcuateb*** function definition present inside AI.py file (this file can be found within the *player* folder).

As part of this project submission, we expect you to give us two things:

Things to Submit: See Table.1 for file name and explanation.

Evaluation: Your code will be graded manually by TA for technical correctness. So make sure your code could be running in the terminal smoothly before making the submission. The final grade of this project is based on how better your designed player can beat the robot in chess.com;

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Question-2.0 Setup on Local Computer (0 Points)

In this question, we use the code provided in the chess-game-AI-master.zip file that implements a minimax search based AI agent for playing the game of chess. As a first step, please use the following instructions to install this repository inside your local computer terminal.

1. Extract all files from the chess-game-AI-master.zip file.
2. cd into the project folder containing the extracted files using, for example, *cd chess-game-AI-project*.
3. Run *pip install -r requirements.txt*
4. In order to run the game, write the command on terminal or cmd: *python3 playchess.py*

Note: You might run into some issues of missing packages, if you do run into them, you can install those missing packages first by running *pip install <package name>.*

To check whether you have successfully set up the environment, you can try playing a game of chess using the GUI application that pops up when run *python3 playchess.py*.

Question-2.1 Understanding the Current Evaluation Function (5 Points)

Go to Line 253 in the chess-game-AI-project/player/AI.py and study the evaluation function implementation in function ***calculateb.*** Try and understand what that evaluation function implementation is trying to do, i.e., what logic is the ***calculateb*** function using to evaluate a non-terminal state.

Once you have understood the logic of **calculateb's** current evaluation function implementation, write down your understanding (in plain English) of the logic of this implementation in submission.pdf.

Question-2.2 Designing a Better Evaluation Function for Chess (25 Points)

Change the function definition of ***calculateb*** in Line 253 so as to design a better chess playing agent.

IMPORTANT NOTE: DO NOT change any other files or functions. You can input print statements or debug statements if you really wanted to, but beyond that, do not change any other files. IMPORTANTLY, DO NOT CHANGE THE DEPTH OF THE MINIMAX TREE FROM 3.

To make things interesting, your AI agent (powered by your unique evaluation function) will be competing against every other team's AI agents in the class. Out of the 25 points available for Question 2:

a) 15 points will be based on evaluation of your code and logic done by the TA and me.
b) 10 points will be based on what rank your AI agent ends up getting as compared to the AI agents designed by the rest of the class.

To get the 15 points, you just have to include a zip file of your updated code.

To get the 10 points, you have to do the following:

1. Test your chess agent against bots on Chess.com, and tell us what bot level you are able to win a majority of games against (assume a best of 5 games scenario).
2. To check whether your current chess agent is competitive against your class peers, create an entry for your team on this Excel sheet.
3. To understand this process of evaluation better, please watch this video.
4. As mentioned in this video, please note down the performance of your AI chess agent in this Google Sheets document.