AI: Connect Four Agent

Brian O'Neill

Western New England University
Springfield, MA, USA
brian.oneill@wne.edu

Course Artificial Intelligence

Programming Language Python, Java

Resource Type Assignment

CS Concepts • VI.9.i Graph Operations / Algorithms

Knowledge Unit Programming Concepts

SYNOPSIS

This assignment allows students to gain experience with AI game-playing algorithms, implementing minimax and alpha-beta pruning and designing a utility function for measuring game states. The assignment uses Connect Four, a relatively simple fully-observable and deterministic game that students are likely to have seen before. Students are responsible only for developing an agent to play the game; the game itself is already implemented and given as part of the student-facing materials. The assignment breaks down the requirements for the two algorithms into smaller chunks in order to make the whole assignment more approachable. We also provide code for Tic-Tac-Toe so that students can apply their code for minimax and alpha-beta pruning to a simpler game where sub-optimal moves will be more obvious, indicating potential bugs in their implementation. The assignment allows for a tournament to be played among all student submissions, potentially awarding extra credit to the winner of the class tournament.

ACM Reference Format:

O'Neill, B. 2022. AI: Connect Four Agent. In ACM *EngageCSEdu*. ACM, New York, NY, USA, 2 pages. https://doi.org.10.1145/3554916

KEYWORDS

Heuristic, game playing, minimax, alpha-beta pruning, connect four, assignment, artificial intelligence.

1 INTRODUCTION

In this assignment, students implement the minimax and alpha-beta pruning algorithms, along with several game-specific functions required by those two algorithms. The domain for this assignment is Connect Four [2], a two-player deterministic and fully-observable game. The game is similar to Tic-Tac-Toe (or Noughts and Crosses), with players trying to place four of their tokens in a row on a 6x7 board. Unlike Tic-Tac-Toe, tokens cannot be placed anywhere on the grid; instead tokens are dropped in a vertical column, falling to the lowest available row.

Students are provided with code that implements Connect Four and provides hooks for an AI player. Therefore, students are only responsible for developing the game-playing algorithms and not also implementing the game itself. Students are also provided with code for Tic-Tac-Toe. This code clarifies some of the expectations of the early Connect Four questions and provides a facility to debug the game-playing algorithms in the later questions.

In the assignment, Question 1 instructs students to develop a simple agent, primarily to ensure that students are familiar with the given interface. In Questions 2-4, students define helper functions necessary for the later minimax and alpha-beta pruning questions. The Tic-tac-toe code includes implementations of these functions, providing additional insight to students about their expected behavior. Question 5 requires students to implement the minimax algorithm. If implemented correctly, students should be able to transfer their code to the Tic-tac-toe agent, which should then win or draw every match. Losses indicate sub-optimal play, meaning there is a bug in their code. Sub-optimal play is much easier to see in Tic-tac-toe, due to its simplicity, than in Connect Four. For Question 6, students implement alpha-beta pruning and develop a heuristic function to measure the utility of a mid-game state. Finally, Question 7 requires students to verbally describe their utility function.

2 ENGAGEMENT HIGHLIGHTS

This assignment uses Meaningful and Relevant Content. When teaching minimax and alpha-beta pruning, it is common to use toy games without a real-world analog (particularly as examples to explain the two algorithms), very simple real-world games (e.g. Tic-tac-toe), or complex games whose rules and strategies are not commonly understood without extensive experience (e.g. chess, go). Tic-tac-toe is too simple for students to find relevant, while chess may be too difficult for students not previously familiar with the game. Using Connect Four splits the difference between these two games – the game and basic strategy are commonly known, and the rules are relatively simple. The complexity of the game (branching factor, maximum depth) falls between Tic-tac-toe and chess.

This assignment also Encourages Student Interaction, suggesting that students test their agents against each other. While establishing a tournament for extra credit may inspire competition and discourage students from playing each other, we actually try to encourage students to play-test their agents against each other in order to see how well the agents are doing. Many students focus on playing against their agents themselves, but find that they are not as strong of a Connect Four player as they thought, losing frequently to agents that are reasonably well-designed. By pitting

ACM EngageCSEdu, August 2022.

ACM ISBN 978-1-4503-9511-3/22/08.
DOI: http://dx.doi.org/ 10.1145/3554916

agents against classmates' agents, students get the chance to integrate their code and discuss the outcomes of their games, potentially strengthening the strategies used in the alpha-beta pruning heuristics.

3 RECOMMENDATIONS

This assignment was developed for an upper-level AI class, with Data Structures as the prerequisite. Game-playing is the second major topic covered in the course, following heuristic search. We announce the assignment after covering minimax and alpha-beta pruning, allowing two to three weeks to complete the assignment.

The assignment contains starter code for both Python and Java. We include both because most of our students have experience in both languages and we want to encourage them to use their preferred language. Adopters could choose to offer only one language without impacting the merits of the assignment.

Faculty adopting this assignment may be tempted to have students implement the game rules, in addition to the AI algorithms. This would substantially increase the scope of the assignment and the time required. Doing so would also mean that students implementing functions to evaluate board state as part of the alpha-beta algorithm would not be able to reference the existing functions to determine winning or losing states.

Adopters looking to replace Connect Four as the game would need to either build a model for their preferred game, such as the model included in this assignment, or require students to build the model, as noted above. In order to keep the assignment at the same level of difficulty, we recommend using another game that is both fully-observable and deterministic, like Connect Four.

4 MATERIALS

The Project 2 Word file is the student-facing assignment instructions. It contains a summary and assigned questions, along with suggestions for how to debug their code for the more challenging parts. Page and section numbers cited in this file are from Russell & Norvig's *AI: A Modern Approach* [1].

The python and java folders contain starter code for the respective languages. The two folders have a parallel structure, so that instructions for the project can be simplified. The differences between the Python and Java code are largely in function and variable naming to correspond to each language's conventions. The Java code is structured into packages, while the Python structure is flat. Some classes that are in separate files for Java are combined into a single file in Python. The following file descriptions are based on the structure for the supplied Java code, though we note the equivalent Python file(s) throughout.

The `c4` package contains two source files and two subpackages. The two source files are ConnectFour.java and IllegalMoveException.java. (The equivalent files are connect4.py and c4exceptions.py.) ConnectFour.java is the main class for the project. Students are provided with code to play a single game or batches of games, with games being output to the console or not at all. Students can modify the code to change the agents being used and to switch between the single game and batch functions. IllegalMoveException.java is an Exception class thrown when an invalid move is received in the game.

The `c4.players` package contains three classes that provide a basis for the students' implementation. Equivalent classes can be found in c4players.py. The ConnectFourPlayer class is an abstract class that any agent must extend. The ConnectFourRandomPlayer class makes random moves. The ConnectFourHumanPlayer class provides an interface for a human to play against an agent using the code provided. Students' agents should be extensions of the ConnectFourPlayer abstract class.

The `c4.mvc` package contains adopts a Model-View-Controller (MVC) pattern to control the game rules, logic, and displays. Students do not need to be familiar with MVC to do this project, as they should not modify any of the code within this package. However, students will need to make use of functions within the ConnectFourModel class (see c4model.py) for their agent to access game information. This class is responsible for tracking the game, including the current state of the board and determining whether the game has reached a win or draw state. The ConnectFourViewInterface (and ConnectFourViewBase abstract class in c4view.py) establish the requirements for displays for the Connect Four game. Two implementations are provided – ConnectFourConsoleView gives text output of the game board and asks for moves from human players using standard input, while ConnectFourSilentView gives no output whatsoever and is intended for batch gameplay. Both Python equivalents are found in c4view.py. While no GUI views are provided, these could easily be added as implementations of the ConnectFourViewInterface or the ConnectFourViewBase abstract class. The ConnectFourController class (c4controller.py) is a standard controller class within the MVC pattern. Explicit observer interfaces (GridObserver and ResultObserver) are included in Java only to allow the View and Controller to receive information from the model. Equivalent functions exist in the Python classes without the interface structure.

The `tictactoe` package, along with the `tictactoe.mvc` and `tictactoe.players` subpackages, provide analogous code to the classes above for Tic-Tac-Toe. (See tictactoe.py, tictactoe_mvc.py, and tictactoe_players.py.) As previously noted, this code is provided as a debugging tool for the minimax and alpha-beta pruning questions.

Finally, the tournament folder includes code for running the extra credit tournament in either Java or Python. This code is not distributed with the student code; rather it is used by the instructor during the grading process to run a class-wide tournament. Tournaments are run entirely within-language; no code is provided to allow a Java agent to play against a Python agent. Adopters must follow the instructions in the comments for these files to make student code accessible and results files readable.

5 REFERENCES

[1] Stuart J. Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

[2] Eric W. Weisstein. "Connect-Four." From *MathWorld*--A Wolfram Web Resource. Retrieved from https://mathworld.wolfram.com/Connect-Four.html