CS255 Artificial Intelligence Coursework 2025–26
Adversarial Search

In this coursework, your task is to implement and evaluate two versions of adversarial search, specifically, the minimax algorithm with and without alpha-beta pruning. Your algorithm will be applied to a game called Connect. This document explains the rules of Connect, the files that you are given (and are required to use), the different elements of the coursework, and how to submit your solution.

Your solution must be written in Python 3 and *must run on the DCS systems using the currently installed Python 3 version*[1] **The deadline for this coursework is 12 noon on Monday 12th January 2026.**

1 Getting Started

To begin, download the zip file from the CS255 Moodle page, which can be found underneath the heading 'Coursework'. This zip file contains several Python files, which together simulate the game environment and players. There is also a readme.txt, which explains the purpose of each file. Please read through the readme.txt carefully. The zip file also contains a Latex template that you must use for your report[2], along with a simple Python script illustrating how to generate plots to include in your report.

The main Python files that you will use in this assignment, namely `player.py`, `board.py` and `runGame.py`, all have comments explaining their functionality. While this document gives an overview of these files, please read the comments in the files carefully.

In particular, it is important to note that **you are NOT permitted to import additional modules** into any file. Including additional imports in your submission will result in mark of zero for the coding aspect of this coursework.

It is also important to note that if your solution has parameters/variables that determine the behaviour of your solution, **you must set these appropriately. The markers will not edit your code to make things work or fine tune parameters to the test cases!**

2 Introducing Connect

Connect is the game environment for this coursework. You are probably familar with the 'Connect 4' version of Connect. The rules of Connect are simple. Each player has a set of pieces, and they take turns placing their pieces into the columns of the board. The first player that arranges their pieces into an uninterrupted line of a given size, wins. This line can be horizontal, vertical or diagonal. Note that for this coursework, unlike 'Connect 4', we will vary the number of pieces

[1]See the DCS user guide: `https://warwick.ac.uk/fac/sci/dcs/intranet/user_guide/`.

[2]If you are not familiar with Latex you might want to use `overleaf.com` (using your University email address).

required to be in a line to create different environments. The board is defined by a given number of rows and columns, which will also be varied within this coursework.

Importantly, the board operates with gravity. On their turn, players select a column in which to place a piece, and their piece will fall to the lowest numbered *empty* row. For example, suppose that a player places their piece in column 4. If spaces $(0, 4)$ and $(1, 4)$ are already occupied[3], then the player's piece will fall to position $(2, 4)$.

3 Overview of the Python Files

This coursework is written in Python 3, and you should use the currently installed Python 3 version on the DCS systems. You can use Python 3 by typing the `python3` command at the prompt, e.g.:

```
dcs-remote-node$ python3 --version
Python 3.9.21
```

The coursework zip contains the following Python files:

3.1 player.py

This is the file where you will implement your solutions. There are methods for both the minimax algorithm without pruning, `getMove()`, and for the minimax algorithm with alpha-beta pruning, `getMoveAlphaBeta()`. This file has a preamble which explains what you are expected to return from each method. Note that you must not change the method names or return types. To implement your solution, you are allowed to make use of (but not to modify) methods in the Board class, found in `board.py`.

3.2 board.py

This file contains the Board class, which maintains the required information to describe the current state of the game environment. The file is commented, and explains how to check for useful information, such as which columns are full. You should read the comments in this file carefully. You may use the methods in this file, but you should not modify it.

3.3 runGame.py

This file is a simple script that will create a Connect game environment, and then run the game. The file is commented to explain how to change the game environment by setting the number of rows, columns, and the number of pieces required to be in a line to win. You can also change whether to use alpha-beta pruning or not. The comments also give examples of the types of environment that will be used to evalate your solution.

[3]Note that (for implementation reasons) we index the *row* and then the *column*.

3.4 game.py

This file contains the Game class, which creates the environment as specified and actually runs the game. It contains the `playGame()` method which cycles through both players and tracks when the game is over. You should not edit this file, and you should not need to consider it beyond providing the necessary arguments in `runGame.py` to create a Game object and play a game.

3.5 randomPlayer.py

This is a simple example of a player class, which selects a random non-full column for the next move. This is provided to help you with (starting) to test your solution. Once you have something sensible, you should use that instead of the random player.

4 Adversarial Search Tasks

Your aim in this coursework is to implement and evaluate two versions of adversarial search, namely, the minimax algorithm with and without alpha-beta pruning. Your algorithm will be applied to the game of Connect introduced above. There are two overall tasks:

- **Task 1:** Minimax without pruning, and
- **Task 2:** Minimax with alpha-beta pruning.

Your solutions for these tasks should be placed within the `getMove()` and `getMoveAlphaBeta()` methods of the `player.py` file as described above. Note that while you can modify `runGame.py` for testing purposes, the only file you submit is `player.py`, and so your submitted code must run with the original version of `runGame.py`.

In addition to implementing these methods, you are also expected to evaluate them and write a brief report documenting this evaluation. You might consider how the methods perform individually and how they compare to each other. Aspects you may discuss include the impact of changing the size of the board, varying the number of pieces required to be in a line to win (for different board sizes), and the overheads of the methods, etc. When performing the evaluation you should consider that different runs may get different results, and so you may wish report details such as the mean and standard deviation for the metrics you discuss.

It is important to note that the code you submit should be the configuration that you want to be marked. In particular, if you have parameters that control how your solution operates, you should set them appropriately since markers will not change parameter values or fine tune your solution.

Your report should be **3 pages long** and **produced with the supplied template**. You are permitted to put references on an additional page, but any content after the page limit will be ignored.

Hints:

- As documented in the code, your implementation should track the number of nodes expanded and the number of times you prune (in the `numExpanded` and `numPruned` variables), which might be useful information to consider in your evaluation. Solutions that do not track these metrics may receive a mark of zero for the coding aspect.
- You should consider how to present your results. It is recommended that you include tables and/or figures (see the template for examples of how to produce these).
- A coarse measure of how long your algorithm takes to run can be found using the `time` command on the DCS systems (e.g., `$time python3 runGame.py`). See the man page for more details.
- You might want to consider the related literature on minimax and its variants. A search on `scholar.google.com` is a reasonable place to start[4].

5 Academic Conduct

You must ensure that you adhere to the University guidelines on cheating and academic conduct — see `https://warwick.ac.uk/fac/sci/dcs/teaching/handbook/examinations#academicmisconduct` and `https://warwick.ac.uk/fac/sci/dcs/teaching/handbook/examinations#aitools`. Also, note that your submission (both the code and report) will be subjected to automated testing for plagiarism.

6 Submission

You should submit two files to Tabula, namely, your python implementation and a PDF of your evaluation.

Your Python file must contain your implementations of the `getMove()` and `getMoveAlphaBeta()` methods. Please rename the `player.py` file to `uniID.py`, where uniID is your University ID number, e.g., `1231231.py`. You should ensure that your submitted file can solve each task and contains no additional imports.

Your evaluation report should be named `uniID-report.pdf`, where uniID is your University ID number. Your report **must** be in PDF format and use the template provided in the coursework zip file.

Please make sure that you submit your coursework before 12 noon on Monday 12th January 2026.

[4]Note that there are several papers on minimax for Connect 4. Reading these for inspiration and context is good academic practise, but please be mindful of your own academic conduct.

7 Mark Scheme

The marks for the coursework will be distributed as follows.

30% Implementation and performance of minimax (without pruning)

20% Implementation and performance of minimax with alpha-beta pruning

50% Evaluation of minimax and minimax with alpha-beta pruning in the context of Connect (i.e., the report), including depth of discussion and quality of analysis

Postscript: A Brief Note About Python

While in this coursework you should use Python, *this is not a test of your Python skills.* You should focus your implementation efforts on the `getMove()` and `getMoveAlphaBeta()` methods. This is, after all, a piece of AI coursework and not an exercise in programming proficiency, and is certainly not a test of your python skills!

The files provided are commented and have been written to be readable, and they are not intended to be particularly efficient or use the most Pythonic way of doing things.

This coursework requires only minimal knowledge of Python and there is no assumption that you have experience of Python before attempting this coursework. The coursework relies primarily on the coding concepts you learnt last year. There are many good basic tutorials (which is all you will need) for Python on the web, but the following give a good introduction to the syntax.

- The 'official' tutorial: `https://docs.python.org/3/tutorial/` (Note that you can change the version using the dropdown at the top of the page.)
- `https://www.w3schools.com/python/python_intro.asp`