

# Assignment 1: Playing Connect 4

## Deadline

**Submission:** 11:59pm, 16 April 2021 (Friday week 6, Sydney time).

*This assignment is worth 12 marks = 12% of your final mark. It is an individual assignment.*

## Marking

The assignment contains two parts: auto-marked and tournament. The auto-marked part is worth 10 marks (10% of your final mark) and the tournament is worth 2 marks (2% of your final mark).

Your mark for the auto-marked part will be your mark from the automatic grading system PASTA, and your mark for the tournament part will be determined by your position in the tournament which we will run after the submission deadline.

## Late submissions policy

- Auto-marked component: late submissions are allowed for up to 3 days late. A penalty of 5% per day late will apply. Assignments more than 3 days late will not be accepted (i.e. will get 0 marks). The day cut-off time is 11:59pm.
- Tournament component: no late submissions are allowed.

## Programming languages

Your implementation can be written in Python, Java, C, C++ or MATLAB. The assignment will be tested using the language versions as described in the “How your program will be run” section below, so it is important that your program can be run in the specified versions.

## Submission

Your assignment must be completed individually using the submission tool PASTA (<https://comp3308.it.usyd.edu.au/PASTA>).

If you are off-campus, you'll need to be connected to the university VPN in order to access the PASTA website. You can read [this page](#) to find out how to connect to the VPN. The students in China impacted by the travel ban should use the [new VPN](#), which is the special VPN for accessing university resources from China. If you are on-campus and connected to the University wireless network, you don't need a VPN to access PASTA.

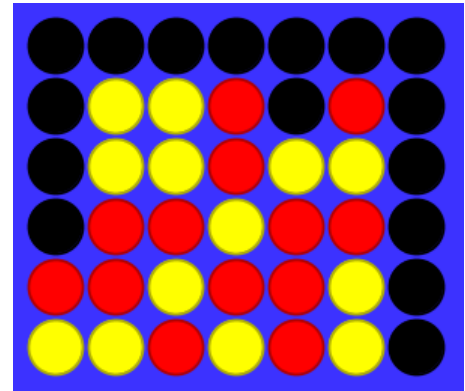
PASTA will allow you to make as many submissions as you wish, and each submission will provide you with feedback on each of the components of the assignment. Your last submission before the assignment deadline will be marked, and the mark displayed on PASTA will be the final mark for your assignment.

## 2. Connect 4

In this assignment, you will implement the minimax search algorithm with and without alpha-beta pruning in order to play a game of Connect 4.

The game of Connect 4 is played on a 6x7 vertical board (6 rows and 7 columns) with 21 red tokens and 21 yellow tokens. Players take turns in putting one of their tokens into the top of one of the columns, where it falls to the bottom-most available slot in that column, thus there are at most seven possible moves at any given point.

The first player to achieve a configuration where four of his/her tokens are lined up (horizontally, vertically or diagonally) wins.



## 3. Tasks

Write a program that, given an initial starting state of the board, and whose turn it is to play, will output the column to play in. Your program will accept inputs to determine whether it will use the regular minimax algorithm, or minimax with alpha-beta pruning. It will also need to accept input of a specified cut-off (maximum search depth).

See the section on “Input and Output” for how your program is expected to behave.

You will need to implement a version that follows strict rules for auto-marking, and you will also have the opportunity to implement your own algorithm to take part in a tournament-style competition.

### The Auto-marked Version

Your auto-marked version of the program will need to follow a strict set of rules in order to be deterministic and automatically testable.

### Search method

Your algorithm will perform a depth-first search (limited by a maximum depth) through the state space. When choosing the next (child) states, examine them in a left-to-right order, so the left-most column (column 0) is considered first, then the next column (column 1) and so on up to the right-most column (column 6).

### Dealing with ties

When choosing the “maximum” or “minimum” node, if there is a tie for the next best node to choose, then choose the node that was examined first (i.e. the left-most one).

### Utility and Evaluation Functions

When implementing this version, you will need use the following (quite unintelligent) utility and evaluation functions:

```
UTILITY(state):
    if red is winner:
        return 10000
```

```

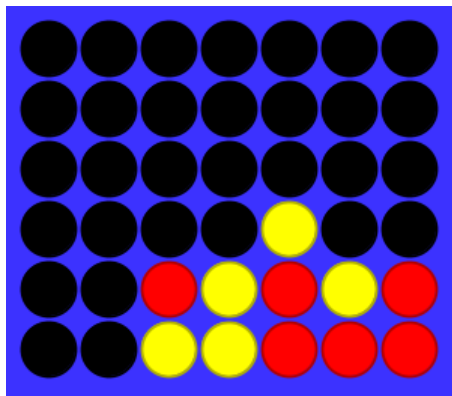
    if yellow is winner
        return -10000

EVALUATION(state):
    return SCORE(state, red player) - SCORE(state, yellow player)

SCORE(state, player):
    return number of tokens of player's colour +
        10 * NUM_IN_A_ROW(2, state, player) +
        100 * NUM_IN_A_ROW(3, state, player) +
        1000 * NUM_IN_A_ROW(4 or more, state, player)

NUM_IN_A_ROW(count, state, player):
    returns the number of times that <state> contains a <count>-in-a-row
    for the given <player>

```



The evaluation function for this board would be calculated as follows (**note that a 3-in-a-row does not count as any 2-in-a-rows**):

$$\begin{aligned} \text{SCORE}(\text{state}, \text{red}) &= 6 + (10 \times 4) + (100 \times 1) \\ &= 146 \end{aligned}$$

$$\begin{aligned} \text{SCORE}(\text{state}, \text{yellow}) &= 5 + (10 \times 3) + (100 \times 1) \\ &= 135 \end{aligned}$$

$$\begin{aligned} \text{EVALUATION}(\text{state}) &= 146 - 135 \\ &= 11 \end{aligned}$$

### The Tournament Version

Your tournament version of the program does not need to be so strictly implemented. You will be free to implement your own algorithm for determining which column to play in. This may be as simple as making up your own version of the evaluation function, or as complicated as writing an entirely different search algorithm.

You must, however, follow the following restrictions:

1. Your algorithm will be timed out after 1 second. If your program does not offer an output before this timeout, your move will be forfeit, and play will return to the other player. *Yes, this timeout is language-agnostic, and yes, it does mean that certain languages will have a slight advantage, however in the past, top performers were written in 'slower' languages.*
2. Your program must be entirely self-contained. It will not be able to interact with the filesystem or the network (no writing files or connecting to the internet).
3. You must still conform to the provided input and output rules, as provided in the "Input and Output" section.

## 4. Input and Output

As your program will be automatically tested, it is important that you adhere to these strict rules for program input and output.

### Input

Your program should be called `ConnectFour`, and will be run from the command line with the following arguments:

1. A string of characters representing the current board state. This string will be in the following format:

```
row0,row1,row2,row3,row4,row5
```

And each row will contain 7 characters, representing the seven columns in the board. The characters will be only `r`, `y` and `.`, representing a red token, a yellow token and a blank space respectively.

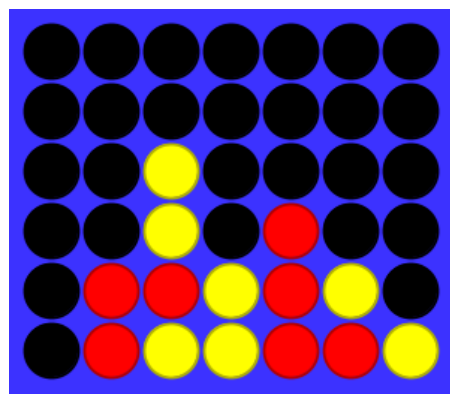
Note: row0 corresponds to the bottom row, and row5 corresponds to the top row.

2. Either "red" or "yellow" to indicate the player who is about to play a piece.
3. Either "M", indicating that your program should use the regular minimax algorithm, or "A" indicating that your program should use minimax with alpha-beta pruning. **This argument will not be provided to your tournament code.**
4. A number representing the maximum depth that the algorithm should search to. **This argument will not be provided to your tournament code.**

For example, if the program is given the following arguments:

```
.ryyrry,.rryry,..y.r...y..... red A 4
```

Then this would be built into the game board below, indicating that it is red's turn to play, and the algorithm should use alpha-beta pruning with a maximum search depth of 4.



### Assumptions about the input

You can assume that all inputs will be sensible, in that the player will only be either "red" or "yellow", the algorithm will only be either "M" or "A", and the depth will be an integer  $> 0$ .

The state provided will be possible to make in a real game (no floating pieces), however it will not necessarily represent a balanced game (e.g. it might be full of red tokens).

## How your program will be run

The following examples show how the program would be run for each of the submission languages, assuming we want to run the above example. For brevity, the game state has been abbreviated to “<state>”, however this would actually be written exactly as in the example above.

Python (version 3.7.0):

```
python ConnectFour.py <state> red A 4
```

Java (version 8):

```
javac ConnectFour.java
java ConnectFour <state> red A 4
```

C (gcc version 6.3.0):

```
gcc -lm -w -std=c99 -o ConnectFour ConnectFour.c *.c
./ConnectFour <state> red A 4
```

C++ (gcc version 6.3.0):

```
g++ -o ConnectFour ConnectFour.cpp
./ConnectFour <state> red A 4
```

MATLAB (version R2018a):

```
mcc -m -o ConnectFour -R -nodisplay -R -nojvm ConnectFour
./run_ConnectFour.sh <MATLAB_install_directory> <state> red A 4
matlab -nodesktop -nosplash -nojvm -nodisplay -r
"try;ConnectFour('<state>','red','A','4');catch
me;disp(me.message);end;quit"
```

**Note:** MATLAB must be run this way (compiled first) to speed up MATLAB running submissions. The arguments are passed to your `ConnectFour` function as strings. For example, the example above will be executed as a function call like this:

```
ConnectFour('<state>','red','A','4')
```

## Output

For your automatically tested version of the program, you will output **two lines only**. The first line will contain the column that the algorithm will play in. This will be a single integer, where 0 represents the left-most column and 6 represents the right-most column.

The second line will be the number of nodes that were examined during the search, where a node is considered examined when you perform the terminal test on it.

As an example, running your code with the example from above:

```
.ryyrry,.rryry,...y.r...y....,.....,..... red A 4
```

Should result in the following output:

```
1
297
```

This indicates that red should play in the second column, and that 297 nodes were examined.

For the tournament version of the code, your program is not required to print the second line. You still need to print the first line (as described above), however this is the only output your program should have.

## 5. Submission Details

This assignment is to be submitted electronically via the PASTA submission system.

Your submission files should be zipped together in a single .zip file and include a main program called ConnectFour. Valid extensions are .java, .py, .c, .cpp, .cc, and .m. Zip only the submission files, not the folder – when your zip file is unzipped there should be only submission files, not a folder with submission files. Only .zip format is accepted; do not use any other format, e.g. .rar or .7z. If your program contains only a single file, then you can just submit the file without zipping it.

Upload your auto-marking submission on PASTA under **Assignment 1**, and your tournament version on PASTA under **Assignment 1 – Tournament**.

Detailed submission instructions:

- Find **Assignment 1** or **Assignment 1 - Tournament** and press “Submit” (the red icon on the right). Then press “Choose File” and attach the file, then press “I accept” after reading the University policy on academic honesty and agreeing with it.
- If you see a message indicating that your code is queued for testing, this means that your code has been uploaded successfully.
- If you see a red error box, you need to fix any problems indicated before your submission will be accepted.
- Once your program has been tested, the page will tell you to refresh for results, so refresh the page. You should see green and red boxes. Each box corresponds to a test; a red box indicates that your program has failed the respective test, a green box indicates that your program has passed the test and a grey box indicates that the test is hidden (you will see the result of the test after the due date).
- If you have red boxes, you can see which tests were not passed by clicking on the name of the task (**Assignment 1** or **Assignment 1 – Tournament**). Correct the errors (go back to your development environment, re-write your code and test it carefully) and then submit again in PASTA.

## 6. Academic Honesty

Please read the University policy on Academic Honesty very carefully:

<https://sydney.edu.au/students/academic-integrity.html>

Plagiarism (copying from another student, website or other sources), making your work available to another student to copy, engaging another person to complete the assignments instead of you (for payment or not) are all examples of **academic dishonesty**. Note that when there is copying between students, both students are penalised – the student who copies and the student who makes his/her work available for copying

The University penalties are severe and include: 1) a permanent record of academic dishonesty on your student file, 2) mark deduction, ranging from 0 for the assignment to Fail for the course and 3) expulsion from the University and cancelling of your student visa. In addition, the Australian Government passed a new legislation last year ([Prohibiting Academic Cheating Services Bill](#)) that makes it a **criminal offence** to provide or advertise academic cheating services - the provision or undertaking of work for students which forms a substantial part of a student's assessment task.

Do not confuse legitimate co-operation and cheating! You can discuss the assignment with another student, this is a legitimate collaboration, but you cannot complete the assignment together – this is an individual assignment and everyone must write their own code.

To detect code similarity in this assignment, we will use the system MOSS which is **extremely good**. If you cheat, the chances that you will be caught are very high. **Be smart and don't risk your future or break the law by engaging in plagiarism and academic dishonesty!**