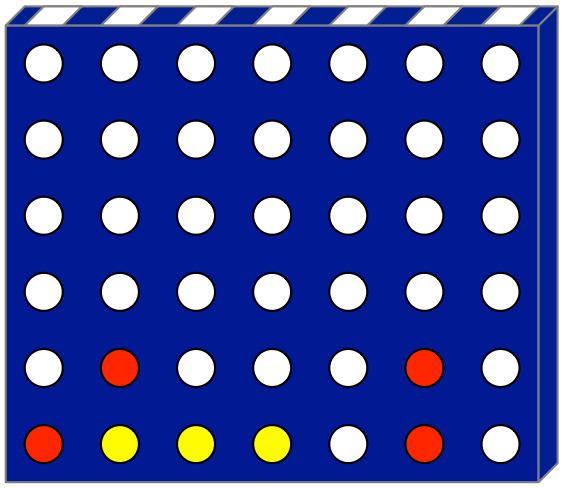
CSC242: Intro to AI
Project 1: Game Playing

This project is about designing, implementing, and evaluating an AI program that plays a game against human or computer opponents. You should be able to build a program that beats you, which is an interesting experience.

The game for this term is Connect-Four. Connect-Four is a game where two players take terms putting markers on a rectangular grid. The first player to form a line of four markers either horizontally, vertically, or diagonally is the winner. This is similar to Tac-Tac-Toe and Gomoku. In fact, these are all variants of what is called an "m, n, k-game" (m rows, n columns, k in a row to win).



As shown in the figure above, the difference is that in Connect-Four, the grid is positioned vertically and rather than placing a marker on a space, a player drops their marker into one of the columns of the grid. The marker comes to rest in the topmost empty cell of that column. Thus yellow can win by dropping their marker into the fifth column. However red cannot play a third marker into the diagonal group at left since the marker will fall to the fifth row. The Wikipedia article has a good description of the game.

Please Note: Connect-Four has been used as a challenge for computers for about as long as there have been computers. If you want to learn anything, avoid searching for information about the game beyond Wikipedia until you have done the basic implementation YOURSELF.

Requirements

1. Develop a program that plays 3x3x3 Connect-Four (a 3x3 grid where you need to place 3 markers in a row horizontally, vertically, or diagonally to win). This is similar to Tic-Tac-Toe except that only the topmost cell of each column is playable on any turn.

    - You must use a state-space search approach to solve the problem.
    - Design your data structures using the formal model of the game.
    - Use the appropriate standard algorithm(s) to select moves.
    - Your program should be able to play quickly and perfectly.

2. Develop a program that plays standard 6x7x4 Connect-Four.

    - You must again use a state-space search approach to solve the problem.
    - If you design this right for Part 1, you will be able to resuse it with *no modification*. How cool is that?
    - Choosing the best move in this game is significantly harder than the smaller game of Part 1. (You should be able to figure out at least an upper bound on how much harder it is.) You will need to use a different standard algorithm to select a move in a reasonable amount of time.
    - Your program should be able to play well. In particular, it should not take too long to choose a move (unless the user tells it to do so).

Your programs should use standard input and standard output to interact with the user. An example is shown on the next page. You are welcome to develop graphical interfaces if you like, but it's not the point of this course.

FYI: If you'd like a nice firsthand (and even somewhat technical) account of building a world-class game-playing program, check out *One Jump Ahead: Computer Perfection at Checkers*, by Jonathan Schaffer (ISBN-13: 978-0387765754).

Example Output

Here's a quick example of what your pogram might look like. A longer transcript is at the end of this document.

```
Connect-Four by Your Name Here
Choose your gamne:
1. Tiny 3x3x3 Connect-Three
2. Wider 3x5x3 Connect-Three
3. Standard 6x7x4 Connect-Four
Your choice? 3
Choose your opponent:
1. An agent that plays randomly
2. An agent that uses MINIMAX
3. An agent that uses MINIMAX with alpha-beta pruning
4. An agent that uses H-MINIMAX with a fixed depth cutoff
Your choice? 4
Depth limit? 8
Do you want to play RED (1) or YELLOW (2)? 2

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4               4
5               5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 6634026 states
  best move: RED@3, value: 0.002976190476190476
Elapsed time: 9.065 secs
RED@3

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4               4
5       X       5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 2
Elapsed time: 14.535 secs
YELLOW@2
```

Suggestions for Success

Start by designing your data structures.

What are the elements of the state-space formalization of a two-player, perfect knowledge, zero sum game? We've seen them in class and in the textbook.

If you're using Java, turn these into interfaces. Hint: An interface doesn't actually have to define any methods, if that is useful. If you're not using Java, you still have the abstract specifications to guide you.

Now, how would you represent the specifics of $m \times n \times k$ Connect-Four using this framework? Design classes that implement the abstract specifications (interfaces) for this specific game (problem domain). You can write test cases for these as you go.

Next: The algorithms for solving the problem of picking a good move are defined in terms of the formal model. So you can implement them using only the abstract interfaces. Write one or more classes that answer the question "What move should I make in this state?" Hint: An agent that picks randomly but legally is not hard, using what you get from the formal model. So start there, but you should know what algorithm you really need for this problem.

Once you have one or more agents for playing the game, write the program that puts the pieces together to generate the gameplay shown in the example transcript. This will get you through Part 1.

What do you predict will happen if you use this program to play the full 6x7x4 version of the game required in Part 2? Try it. You should know from class and from the textbook what algorithm you need if you haven't implememted it already. This may require some "additional information," which you will need to decide on and then add to your game-specific classes (because it's specific to this problem, right?).

The great thing about using the state-space search framework is that you can try different algorithms using the same representation of the problem. Even better, you can use the same algorithms to play different games just by changing the game-specific implementation of the abstract interfaces derived from the formal model of state-space search. And if the descriptions of the games were themselves machine-readable. . . general game playing perhaps?

Project Submission

Your project submission MUST include the following:

1. A README.txt file or PDF document describing:
   (a) Any collaborators (see below)
   (b) How to build your project
   (c) How to run your project's program(s) to demonstrate that it/they meet the requirements
2. All source code for your project. Eclipse projects must include the project settings from the project folder. Non-Eclipse projects must include a `Makefile` or shell script that will build the program per your instructions, or at least have those instructions in your README.txt.
3. A completed copy of the submission form posted with the project description. Projects without this will receive a grade of 0. If you cannot complete and save a PDF form, submit a text file containing the questions and your (brief) answers.

Writeups other than the instructions in your README and your completed submission form are **not** required.

We must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better grade your will be.** It is your job to make both the building and the running of programs easy and informative for your users.

Programming Practice

Use good object-oriented design. No giant `main` methods or other unstructured chunks of code. Comment your code liberally and clearly.

You may use Java, Python, or C/C++ for this project. I recommend that you use Java. Any sample code we distribute will be in Java. Other languages (Haskell, Clojure, Lisp, *etc.*) by arrangement with the TAs only.

You may **not** use any non-standard libraries. Python users: that includes things like NumPy. Write your own code—you'll learn more that way.

If you use Eclipse, make it clear how to run your program(s). Setup Build and Run configurations as necessary to make this easy for us. Eclipse projects with poor configuration or inadequate instructions will receive a poor grade.

Python projects must use Python 3 (recent version, like 3.6.x). Mac users should note that Apple ships version 2.7 with their machines so you will need to do something different.

If you are using C or C++, you should use reasonable "object-oriented" design not a mish-mash of giant functions. If you need a refresher on this, check out the C for Java Programmers guide and tutorial. You **must** use "`-std=c99 -Wall -Werror`" **and** have a clean report from `valgrind`. Projects that do not follow both of these guidelines will receive a poor grade.

Late Policy

Late projects will **not** be accepted. Submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

Collaboration Policy

You will get the most out of this project if you write the code yourself.

That said, collaboration on the coding portion of projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC242.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit code on the group's behalf in addition to their writeup. Other group members should submit only a README indicating who their collaborators are.

- All members of a collaborative group will get the same grade on the project.

Academic Honesty

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.

Example Output

Here's the full transcript of the game shown earlier. I played yellow (second to move). I felt like I was playing defense, but the heuristic estimates starting trending in my favor. Then on my sixth move I messed up and left the program with an easy win (score 1.0), which it found easily (about three seconds).

Note the display of useful information about what the program was doing, such as elapsed time, number of states, utility of the chosen move (estimated utility, in this case), and so on.

```
Connect-Four by Your Name Here
Choose your gamne:
1. Tiny 3x3x3 Connect-Three
2. Wider 3x5x3 Connect-Three
3. Standard 6x7x4 Connect-Four
Your choice? 3
Choose your opponent:
1. An agent that plays randomly
2. An agent that uses MINIMAX
3. An agent that uses MINIMAX with alpha-beta pruning
4. An agent that uses H-MINIMAX with a fixed depth cutoff
Your choice? 4
Depth limit? 8
Do you want to play RED (1) or YELLOW (2)? 2

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4               4
5               5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 6634026 states
  best move: RED@3, value: 0.002976190476190476
Elapsed time: 9.065 secs
RED@3

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4               4
```

```
5       X       5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 2
Elapsed time: 14.535 secs
YELLOW@2

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4               4
5     O X       5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 6489398 states
  best move: RED@3, value: 0.004464285714285714
Elapsed time: 9.450 secs
RED@3

  0 1 2 3 4 5 6
0               0
1               1
2               2
3               3
4       X       4
5     O X       5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 3
Elapsed time: 3.682 secs
YELLOW@3

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O       3
4       X       4
5     O X       5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 6374097 states
  best move: RED@4, value: 0.00595238095238095
```

```
Elapsed time: 9.929 secs
RED@4

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O       3
4       X       4
5     O X X     5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 5
Elapsed time: 4.780 secs
YELLOW@5

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O       3
4       X       4
5     O X X O   5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 6212168 states
  best move: RED@4, value: 0.004464285714285716
Elapsed time: 10.301 secs
RED@4

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O       3
4       X X     4
5     O X X O   5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 4
Elapsed time: 4.989 secs
YELLOW@4

  0 1 2 3 4 5 6
0               0
1               1
2               2
```

```
3       O O     3
4       X X     4
5     O X X O   5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 5190492 states
  best move: RED@2, value: 0.0014880952380952363
Elapsed time: 8.964 secs
RED@2

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O O     3
4     X X X     4
5     O X X O   5
  0 1 2 3 4 5 6
Next to move: YELLOW

Your move [column 0-6]? 5
Elapsed time: 7.752 secs
YELLOW@5

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O O     3
4     X X X O   4
5     O X X O   5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 4648099 states
  best move: RED@1, value: -0.004464285714285716
Elapsed time: 8.279 secs
RED@1

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O O     3
4     X X X O   4
5   X O X X O   5
  0 1 2 3 4 5 6
Next to move: YELLOW
```

```
Your move [column 0-6]? 5
Elapsed time: 5.356 secs
YELLOW@5

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O O O   3
4     X X X O   4
5   X O X X O   5
  0 1 2 3 4 5 6
Next to move: RED

I'm thinking...
  visited 1798769 states
  best move: RED@1, value: 1.0
Elapsed time: 3.182 secs
RED@1

  0 1 2 3 4 5 6
0               0
1               1
2               2
3       O O O   3
4   X X X X O   4
5   X O X X O   5
  0 1 2 3 4 5 6
Next to move: YELLOW

Winner: RED
Total time:
       RED: 59.168 secs
    YELLOW: 41.093 secs
```