CSC242: Intro to AI
Project 1: Game Playing

This project is about designing an implementing 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 Reversi, also known as Othello.

Reversi is an easy game to describe, which makes it easy to learn for humans and easy to program for computers. But it's also difficult to play well, at least for humans, for a couple reasons. One is that all the pieces are the same, unlike a game like chess. And unlike a game like checkers (draughts), the number of pieces on the board increases rather than decreases. Pieces also change color between dark and light during the game. These factors may make it harder for humans to track the evolution of the game, to quickly evaluate positions, and to memorize positions and strategies for playing from them. You'll find out whether that's true for computers also.

Please Note: There is a long history of computer programs that play Reversi/Othello. Wikipedia says that it was one of the first arcade games developed by Nintendo and was available on a home game console as early as 1980. If you want to learn anything, avoid searching for information about the game beyond the Wikipedia page linked above until you have done the basic implementation **YOURSELF**.

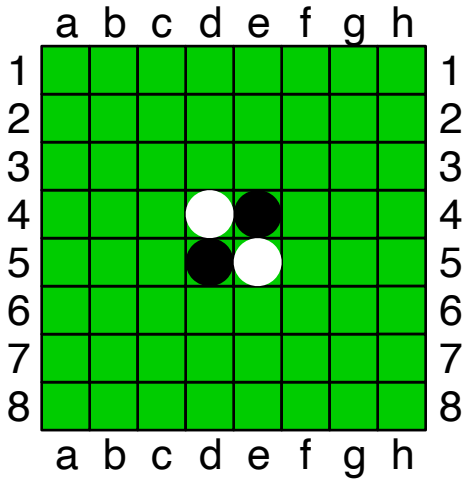

Figure 1: Initial arrangement of pieces for standard 8x8 Reversi

Rules of Reversi (Othello)

Board and Pieces

- The game is played on a rectangular board divided into squares as shown in Figure 1. The standard size is 8x8.
- Columns are labelled with letters starting with “a”. Rows are labelled with numbers starting with 1. Squares are referred to by column and row, from in “a1” to “h8” (on an 8x8 board).
- The pieces are disks with two sides, corresponding to the two players. Traditionally the pieces are dark on one side and light on other, and we refer to the two players as “dark” and “light” also.
- The initial configuration of the board for standard 8x8 reversi (Othello rules) is shown in the figure. Two pieces are placed with their dark side up at locations d5 and e4, and two pieces are placed with the light side up at locations d4 and e5.

Moves

When it is their move, a player **must** place a piece with their color up such that there is a line of one of more pieces of the opponent's color between the new piece and one of the player's own pieces already on the board. That is, each move **must** "trap" a line of the opponent's pieces between the player's own pieces, including the piece being played.

The dark player **must** play pieces with the dark side up trapping a line of light pieces between the new piece and an existing dark piece. The light player **must** do the same with a light piece trapping a line of dark pieces.

The line of one or more trapped pieces may be horizontal, vertical, or diagonal.

For example, from the initial configuration of the board, dark has four possible moves. The first is at c4, trapping the light piece at d4 horizontally between the dark piece at e4 and the new piece. The other moves are at d3 (trapping d4 vertically), at e6 (trapping e5 vertically), and at f5 (trapping e5 horizontally). I suggest that you draw this on a piece of paper or a whiteboard to see what's going on.

After playing their piece, the player flips over the line of trapped pieces so that they all show the player's own color. That is, the dark player plays a piece dark side up, trapping a line of light pieces. They then flip the trapped pieces to be dark side up. Similarly for the light player.

For example, if dark plays at c4, then the light piece at d4 is flipped to dark, resulting in a line of dark pieces from c4 through d4 to e4. This is shown in Figure 2.

Note that a player **may capture more than one line** of their opponent's pieces in a single move. All of the opponent's pieces that are "between" the newly-placed piece and any of the player's other pieces (in a line horizontally, vertically, or diagonally) are flipped.

Players take turns moving.

The player with the dark pieces moves first.

If a player cannot make a legal move, they **must** "pass" (forfeit their turn and not place a piece). However a player may only pass **when they have no legal moves**.

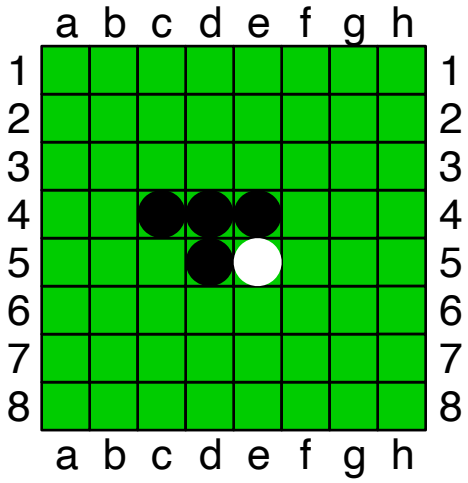

Figure 2: Board position after dark play at c4

End of Game

The game ends when neither player can make a legal move.

This can happen when:

- The board is completely full.
- The board is not full but neither player make a legal move (that is, play a piece that traps a line of the opponent's pieces). Wikipedia has some examples of this FYI.

At the end of the game, the player with the most pieces on the board (the most pieces with their color up) is the winner.

Ties (draws) are possible.

Requirements

1. Develop a program that plays Reversi on a 4x4 board. The four initial pieces go in the middle of the board, as shown in Figure 3. This is not a hard game, but it is simple enough that you can see how your program is playing. And you can probably play pretty well yourself, even if you've never played Reversi.

    - You **must** use an adversarial state-space search approach to solving the problem.
    - Design your data structures using the formal model of the game. We **must** see classes corresponding to the elements of the formal model (see below for more about this).
    - You **must** use the appropriate standard algorithm to select *optimal* moves. Do not use algorithms that make *heuristic* decisions for this part of the project.
    - Your **must** be able to search the entire tree for 4x4 Reversi and hence your program **must** be able to play perfectly and quickly (on modern computers).
    - You may, if you want, also play larger boards with your optimal player. It should require almost no additional code. You should gain an appreciation for the complexity of search problems if you try it.
    - Your program **must** validate the user's input and only allow the user to make legal moves. This is not as hard as it may seem. Think about it. Your program knows a lot about what is possible and what isn't.

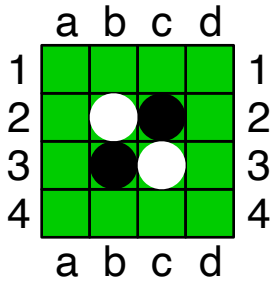

Figure 3: Initial board configuration for 4x4 Reversi

2. Develop a program that plays standard 8x8 Reversi. This can be a separate program, or you can have a single program that asks the user which game to play.

    - You **must** again use an adversarial state-space search approach to solve the problem.
    - If you design this right for Part 1, you will be able to reuse it with *almost no work*. In fact, you will be able to adapt your program to *any* two-player, perfect knowledge, zero-sum game with very little work. 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 **must use heuristic MINIMAX with alpha-beta pruning** for this part of the project.
    - Your program should be able to play well. In particular, it should not take too long to choose a move. Your program should give the user a chance to specify the search limit, using one or more of the ideas described in the textbook and discussed in class.
    - Again, you may also try playing different board sizes with this player if you want.

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 that's not the point of this course and will not be considered in grading.

Here's a quick example of what your program might look like when it runs. A transcript of a complete game with some interesting situations is at the end of this document.

```
Reversi by George Ferguson
Choose your game:
1. Small 4x4 Reversi
2. Medium 6x6 Reversi
3. Standard 8x8 Reversi
Your choice? 1
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 and a-b pruning
Your choice? 1
Do you want to play DARK (X) or LIGHT (O)? x

  a b c d
1         1
2   O X   2
3   X O   3
4         4
  a b c d
Next to play: DARK/X

Your move (? for help): a2
Elapsed time: 9.533 secs
X@a2

  a b c d
1         1
2 X X X   2
3   X O   3
4         4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@c1

  a b c d
1     O   1
2 X X O   2
3   X O   3
4         4
  a b c d
Next to play: DARK/X
```

How To Do This Project

I will assume that you are using Java for this. Why wouldn't you use Java for a program that involves representations of many different types of objects with complicated relationships between them and algorithms for computing with them? Seriously. That's the kind of thing that Java was designed for. If you want to ignore my advice and use Python, well ok, but it has to be well-designed and object-oriented, not a mishmash of lists and dictionaries. See "Programming Practice," below.

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. They can be used to formalize any game, not just Reversi.

In Java, you would probably turn these into interfaces. Python now has some things similar to interfaces, but Python is philosophically opposed to careful specfication of behavior ("duck typing"), which is one reason why I don't recommend Python for large projects.

Now, how would you represent the specifics of Reversi using this framework? Don't forget to think about supporting different board sizes and/or starting configurations, per the requirements. Design classes that implement the abstract specifications (interfaces) for this specific game (problem domain). You can write simple test cases for these as you go and include them in each class' `main` method.

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 "Given a state, what is the best move (for the player whose turn it is to move in that state)?" Usually this involves generating all the possible moves in a state, so you should probably start there, and then figure out how to pick the *best* move. 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 algorithms you really need for this problem.

Once you have one or more of these "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 version of the game required in Part 2? Try it. You should know from class and from the textbook

what algorithms and techniques you need to make a **heuristic** decision about what to do. This may require some "additional information," which you will need to decide on and then add to your game-playing agent(s). The algorithm(s) apply to any game. The "additional information" is specific to a given game, or more precisely, to a specific way of playing a specific game. You might want to try several ways.

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?

Additional Requirements and Policies

The short version:

- You may **only** use Java or Python. I **STRONGLY** recommend Java.
- You **must** use good object-oriented design (yes, even in Python).
- There are other language-specific requirements detailed below.
- You must submit a ZIP including your source code, a README, and a completed submission form by the deadline.
- You **must** tell us how to build your project in your README.
- You **must** tell us how to run your project in your README.
- Projects that do not compile will receive a grade of **0**.
- Projects that do not run or that crash will receive a grade of **0** for whatever parts did not work.
- Late projects will receive a grade of **0** (see below regarding extenuating circumstances).
- **You will learn the most if you do the project yourself**, but collaboration is permitted in groups of up to 3 students.

Detailed information follows. . .

Programming Requirements

- You may use Java or Python for this project.
  - I **STRONGLY** recommend that you use Java.
  - Any sample code we distribute will be in Java.
  - Other languages (Haskell, Clojure, Lisp, *etc.*) only by prior arrangement with the instructor.
- You **must** use good object-oriented design.
  - You **must** have well-designed classes (and perhaps interfaces).
  - Yes, even in Python.
- No giant `main` methods or other unstructured chunks of code.
  - Yes, even in Python.
- Your code should use meaningful variable and function/method names and have plenty of meaningful comments.
  - I can't believe that I even have to say that.
  - And yes, even in Python.

Submission Requirements

You **must** submit your project as a ZIP archive containing the following items:

1. The source code for your project.
2. A file named `README.txt` or `README.pdf` (see below).
3. A completed copy of the submission form posted with the project description (details below).

Your README **must** include the following information:

1. The course: "CSC242"
2. The assignment or project (*e.g.*, "Project 1")

3. Your name and email address
4. The names and email addresses of any collaborators (per the course policy on collaboration)
5. Instructions for building and running your project (see below).

The purpose of the submission form is so that we know which parts of the project you attempted and where we can find the code for some of the key required features.

- **Projects without a submission form or whose submission form does not accurately describe the project 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.

Project Evaluation

You **must** tell us in your README file how to build your project (if necessary) and how to run it.

Note that we will **not** load projects into Eclipse or any other IDE. We **must** be able to build and run your programs from the command-line. If you have questions about that, go to a study session.

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 your grade will be.** It is **your** job to make the building of your project easy and the running of its program(s) easy and informative.

For **Java** projects:

- The current version of Java as of this writing is: 19.0.1 (OpenJDK)
- If you provide a `Makefile`, just tell us in your README which target to make to build your project and which target to make to run it.
- Otherwise, a typical instruction for building a project might be:

```
javac *.java
```

Or for an Eclipse project with packages in a `src` folder and classes in a `bin` folder, the following command can be used from the `src` folder:

```
javac -d ../bin `find . -name '*.java'`
```

- And for running, where `MainClass` is the name of the main class for your program:

  ```
  java MainClass [arguments if needed per README]
  ```

  or

  ```
  java -d ../bin pkg.subpkg.MainClass [arguments if needed per README]
  ```

- You **must** provide these instructions in your README.

For **Python** projects:

I strongly recommend that you **not** use Python for projects in CSC242. All CSC242 students have at least two terms of programming in Java. The projects in CSC242 involve the representations of many different types of objects with complicated relationships between them and algorithms for computing with them. That's what Java was designed for.

But if you insist...

- The latest version of Python as of this writing is: 3.11.1 (python.org).
- You must use Python 3 and we will use a recent version of Python to run your project.
- You may **NOT** use any non-standard libraries. This includes things like NumPy or pandas. Write your own code—you'll learn more that way.
- We will follow the instructions in your README to run your program(s).
- You **must** provide these instructions in your README.

For **ALL** projects:

We will **NOT** under any circumstances edit your source files. That is your job.

Projects that do not compile will receive a grade of 0. There is no way to know if your program is correct solely by looking at its source code (although we can sometimes tell that is incorrect).

Projects that do not run or that crash will receive a grade of 0 for whatever parts did not work. You earn credit for your project by meeting the project requirements. Projects that don't run don't meet the requirements.

Any questions about these requirements: go to study session **BEFORE** the project is due.

Late Policy

Late projects will receive a grade of 0. You **must** 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

I assume that you are in this course to learn. You will learn the most if you do the projects **yourself**.

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

- Teams of no more than 3 students, all currently taking CSC242.
- Any team member **must** be able to explain anything they or their team submits, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the team should submit code on the team's behalf in addition to their writeup. Other team members **must** submit a README (only) indicating who their collaborators are.
- All members of a collaborative team will get **the same grade** on the project.

Academic Honesty

I assume that you are in this course to learn. You will learn nothing if you don't do the projects **yourself**.

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.

Full Game Transcript

Here's the transcipt of a full game of 4x4 Reversi. I played dark against a computer player that picks randomly among its legal moves. I managed to win, but I didn't think I would even at the very end. It's an interesting game. . .

Note 1: When the computer played at a3 (their third move), it flips the pieces at both b2 (diagonal from a3) and b3 (horizontal from a3).

Note 2: On my second-to-last move, I have no legal moves. So I have to pass. But the game is not over until we both don't have any legal moves. Once they play at d3, then I have a legal move at d2. And it wins me the game by flipping pieces both horizontally and vertically.

```
Reversi by George Ferguson
Choose your game:
1. Small 4x4 Reversi
2. Medium 6x6 Reversi
3. Standard 8x8 Reversi
Your choice? 1
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 and a-b pruning
Your choice? 1
Do you want to play DARK (X) or LIGHT (O)? x

  a b c d
1         1
2   O X   2
3   X O   3
4         4
  a b c d
Next to play: DARK/X

Your move (? for help): ?
Enter moves as COLUMN then LETTER.
For example, "a1" is the top left corner.
Enter "pass" (without the quotes) if you have no legal moves.

Your move (? for help): a2
Elapsed time: 9.533 secs
X@a2

  a b c d
1         1
```

```
2 X X X   2
3   X O   3
4         4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@c1

  a b c d
1     O   1
2 X X O   2
3   X O   3
4         4
  a b c d
Next to play: DARK/X

Your move (? for help): d4
Elapsed time: 154.088 secs
X@d4

  a b c d
1     O   1
2 X X O   2
3   X X   3
4       X 4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@c4

  a b c d
1     O   1
2 X X O   2
3   X O   3
4     O X 4
  a b c d
Next to play: DARK/X

Your move (? for help): d4
location is occupied: d4
Try again

Your move (? for help): 4d
invalid location: 4d
Try again

Your move (? for help): a1
```

```
invalid move: X@a1
Valid moves are: [X@b4, X@d1, X@d2, X@d3]
Try again

Your move (? for help): d1
Elapsed time: 53.313 secs
X@d1

  a b c d
1     O X 1
2 X X X   2
3   X O   3
4     O X 4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@a3

  a b c d
1     O X 1
2 X O X   2
3 O O O   3
4     O X 4
  a b c d
Next to play: DARK/X

Your move (? for help): a1
Elapsed time: 30.926 secs
X@a1

  a b c d
1 X   O X 1
2 X X X   2
3 O O X   3
4     O X 4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@b1

  a b c d
1 X O O X 1
2 X O X   2
3 O O X   3
4     O X 4
  a b c d
Next to play: DARK/X
```

```
Your move (? for help): a4
Elapsed time: 12.071 secs
X@a4

  a b c d
1 X O O X 1
2 X O X   2
3 X X X   3
4 X   O X 4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@b4

  a b c d
1 X O O X 1
2 X O X   2
3 X O X   3
4 X O O X 4
  a b c d
Next to play: DARK/X

Your move (? for help): d2
invalid move: X@d2
Valid moves are: []
Try again

Your move (? for help): pass
Elapsed time: 12.137 secs
X@pass

  a b c d
1 X O O X 1
2 X O X   2
3 X O X   3
4 X O O X 4
  a b c d
Next to play: LIGHT/O

I'm picking a move randomly...
Elapsed time: 0.000 secs
O@d3

  a b c d
1 X O O X 1
2 X O O   2
3 X O O O 3
4 X O O X 4
```

```
  a b c d
Next to play: DARK/X

Your move (? for help): d2
Elapsed time: 11.831 secs
X@d2

  a b c d
1 X O O X 1
2 X X X X 2
3 X O O X 3
4 X O O X 4
  a b c d
Next to play: LIGHT/O

Winner: DARK/X
Total time:
    DARK/X: 283.900 secs
   LIGHT/O: 0.000 secs
```