COMP 424 Final Project Game: *Reversi Othello*

Course Instructor: David Meger and Golnoosh Farnadi
Due Date: Friday Dec 1st, 2024, 8:59PM EST
Source for this file: `https://www.overleaf.com/read/vnygbjryrxrt#7b70cb`

This document describes the game that we will use for the project and your AI objectives, as well as the report instructions. In fact, this file itself will be the template for you to copy to start writing your report (if you want to use another tool like Word, that's OK, but make the format match). The starter code, and programming instructions for your agent can be found at:
`https://github.com/dmeger/COMP424-Fall2024`.

1. Goal

The game playing AI algorithms discussed in class are some of the most important ones to know. Like almost all of AI, deploying them effectively in practice requires a balancing between computation time, accuracy, precision and breadth of information computed, all mixed with a healthy amount of trial-and-error that defines the position of *AI Scientist*. Let's explore these ideas in the context of a fun, large-scale problem. This year we will be working on a game called ***Reversi Othello!***

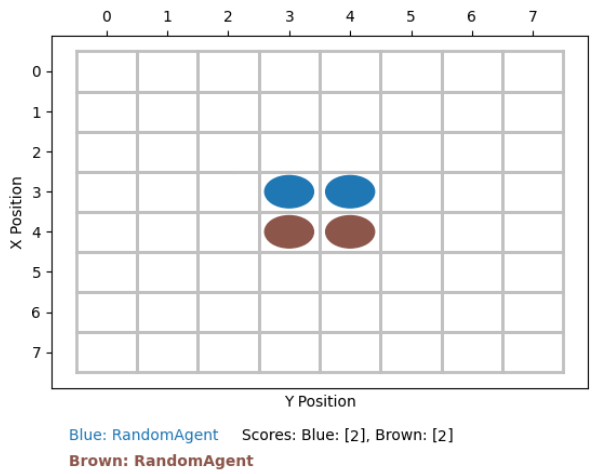


Figure 1: Gameboard

2. Reversi Othello: 2-Player Turn-Based Strategy Game

Reversi Othello is a 2-player turn-based strategy game played on an M $\times$ M board (traditionally 8 $\times$ 8). Players take turns placing discs on the board with the objective of capturing the opponent's discs by surrounding them. M must be an even number, and for evaluation,

we will focus on games where M ranges from 6 to 12. The goal is to have the majority of discs on the board by the end of the game.

2.1 Setup

At the start of the game, each player has two discs placed in the four central squares of the board. Player A uses brown discs, and Player B uses blue discs. Players take turns placing discs, starting with Player A. Discs can be placed in any empty square, provided they flank one or more of the opponent's discs horizontally, vertically, or diagonally. Once a disc is placed, all of the opponent's discs that are flanked by the current player's discs (in any direction) are captured and flipped to the current player's color.

2.2 Objective

The game ends when neither player can make a valid move, which happens when either all squares on the board are filled, or on a partially filled board where no move leads to flipping. The player with the majority of discs on the board at the end of the game is declared the winner, receiving 1.0 points. In the case of a tie, both players are awarded 0.5 points.

2.3 Playing the game

For each turn, a player must provide the row and column, in format (r,c), coordinates of the square where they wish to place their disc. The game engine will automatically flip the opponent's discs that are flanked by the current player's discs in any direction (horizontal, vertical, or diagonal). If a player cannot make a valid move, their turn is automatically passed by the game logic, and the opponent will continue.

3. Assignment Details

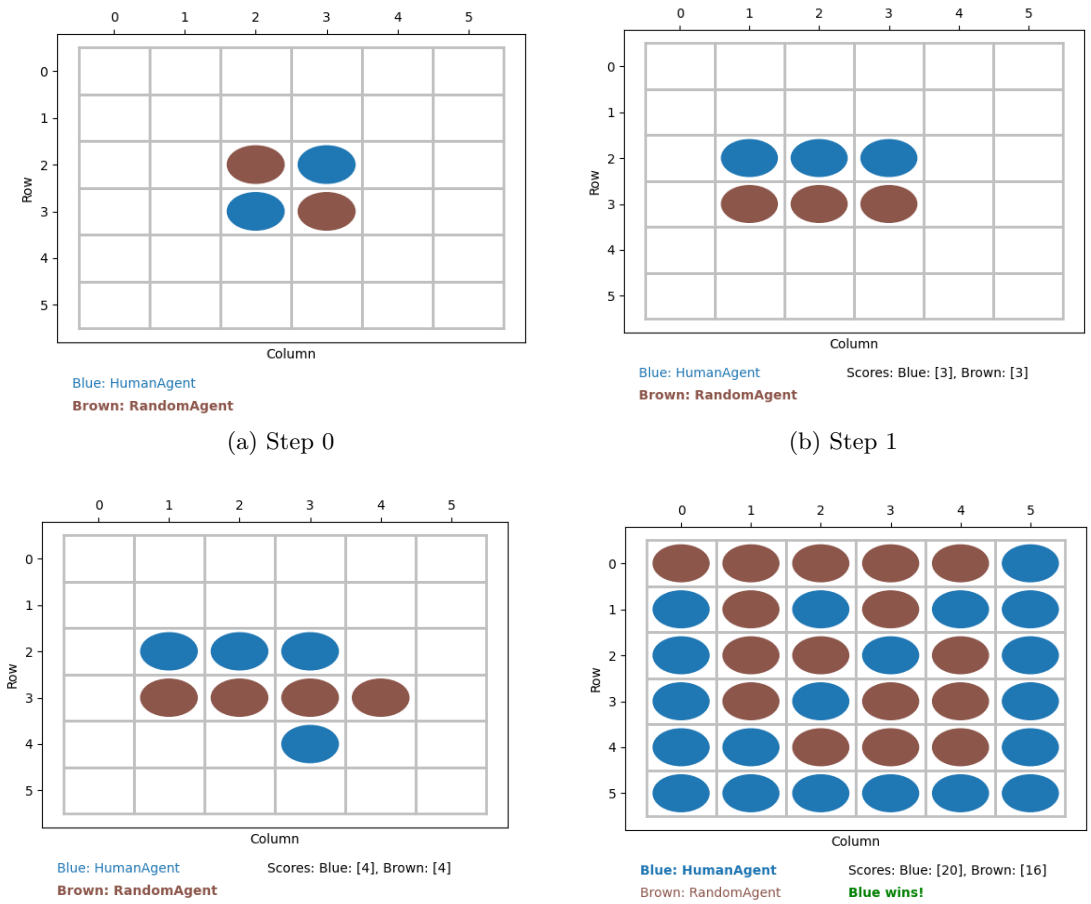
In this final project, your task is to develop an *agent* to play the *Reversi Othello* game. We have developed a minimalistic game engine in Python, which you will extend to add your own *agents*.

3.1 Pre-requirements

This project you will need to implement agent that plays Reversi Othello using Python. Specifically, we strongly recommend to brush up Python 3 fundamentals before writing your code.

3.2 Implementing your own agent

You need to write your own *agent* and submit it for the class project. Detailed instructions of various parts of the game is available in the **README.md** file, visible on the main Github website. Follow the steps there to implement and test your own agent. Be very precise about the instructions regarding your submitted student agent, because we need to be able to run your code automatically.
Important: You should not modify any other files apart from `student_agent.py` and your `authors.yaml` files, as we will copy your file into a "fresh" code checkout that includes all



(a) Step 0

(b) Step 1

(c) Step 2

(d) Step 3

Figure 2: A sample game between two agents (HumanAgent and RandomAgent) on a 6x6 chessboard. The board is initialized (Step 0) with the symmetrical piece placements. HumanAgent then places blue $(2, 1)$ and the RandomAgent places brown on $(3, 1)$ (Step 1). The HumanAgent places a blue on $(4, 3)$ then the RandomAgent places brown on $(3, 4)$ (Step 2). Fast forward and the HumanAgent wins the game with an example endgame shown (Step 3).

other parts of the game. Therefore, functionality changes you make in the simulator or world itself will be lost, and if this causes your code to crash, you will not receive performance grades. The provided helpers module has most of what we expect you to need related to interacting with the game, and you can also look at the sample agents to get started. Any further variables or functions you need should also be created within the student agent file. In the event that any update to the provided game code happens while the project is live, the TAs will announce the necessary steps in Ed to pull the starter code.

3.3 Analyzing and Improving Your Agent

Please read ahead and understand the report sub-sections in advance. If you simply win the tournament but have no idea what your agent is doing, you can get a full performance grade but still do poorly on the project. You should save at least a quarter of your overall effort simply to run the agents you've collected up to that point with printing, reporting or other diagnostics enabled so that you have a good idea why things work as they do and you have good things to say in the report component. Understanding and being able to clearly describe and compare the key points of these AI methods is also the key learning outcome of the project.

4. Report

You are required to write a report with a detailed explanation of your approach and reasoning. The report must be a typed PDF file, and should be free of spelling and grammar errors. The suggested length is between 4 and 8 pages, but the most important constraint is that the report be clear and concise. You can use the source of this document [1] as a starting point for your report by creating an overleaf account starting from the same template and replacing the content as appropriate. Any other way to prepare it is also OK, but make the formatting look the same. The report must include the following required components:

- What is the strongest algorithm you found to play Reversi? A brief overall motivation or executive summary for the best approach. Skip details but give the reader and overview of what parts of the method are most important to achieve strong performance, what play quality you think you achieved and how you came to these conclusions (math/algorithm analysis, iterative design, reading sources, etc).

- A detailed explanation of your agent design, including a re-statement of any relevant general theoretical elements and algorithms (with citations), as well as specific details about how each of these maps to our game. Do not copy your code into this section, rather use English to describe code elements and data structures where they're relevant (roughly 2 pages).

- Analyze your agent's quantitative performance using criteria we mentioned in class. For each, state a numerical quantity or formula and give a 3-4 line text explanation (overall 1 page):

    1. What **depth** (a.k.a. look-ahead) level does your agent achieve board? Is it the same for all branches, or deeper in some cases than others? List elements of your approach that dealt with search depth.
    2. What **breadth** does your agent achieve? That is, how many moves do you consider at each level of play? Is this the same or different for the max and min player? Does your approach address move ordering, pruning or depth-first elements that may reduce the breadth?

1. https://www.overleaf.com/read/vnygbjryrxrt7b70cb

3. What impact does board size have on your method (in particular, its look-ahead and search breadth, as listed above). Did you customize or analysis any method elements based on the board's size?
4. List the heuristics, pruning methods and move ordering approaches you tried, if any. Comment on the impact of each and why some were stronger than others.
5. Predict your win-rates in the evaluation, against: (i) The random agent, (ii) Dave (an average human player), and (iii) your classmates' agents.

- A summary of the advantages and disadvantages of your approach, expected failure modes, or weaknesses of your program. (half page)
- A brief description of how you would go about further improving your player (e.g. by introducing other AI techniques, changing internal representation etc.) These can be ideas you conceived but ran out of time to implement (one page)
- (Conditional) If you have used a significant input source including an LLM, Stack Overflow, past years' 424 code or a tutorial on game playing agents, you must describe that input in up to 1 extra page. For ChatGPT, provide the first parts of the prompt you used and outline how it was created. In all cases, describe what elements of your solution are exact duplicates of the input source and describe the changes you've made on top of these.

5. Submission

First, fill your and your team members (maximum 2 students per team) details in `authors.yaml` file in the repository. Then, **the first student listed in the authors.yaml should submit** to the My Courses folder. Please check the formatting and info in that file very carefully and do not submit the same project twice. If you do, we cannot ensure each partner will receive the same grade.

Do not create a zip, just select the multiple files:

- Report as a PDF file (not the latex source)
- student_agent.py
- authors.yaml

You can make multiple submissions up to the deadline, but in this case we will be very strict about not accepting late code because we'll get the automated grading process running immediately after the deadline.

6. Grading Scheme

50% of the project grade will be allotted for performance in the tournament, and the other 50% will be based on your report.

6.1 Tournament Grading Scheme

The top scoring agent(s) will receive full marks for performance, other top agents in the tournament will will receive marks according to a linear interpolation scheme based on the number of wins/losses they achieve. To get a passing grade on the performance portion, your agent must beat the random player.

The performance grade will be based on your agent's strength rating compared to other students and baselines implemented by the TAs. We will use several phases to group your agents by strength. A first pass is whether you can beat the random agent nearly 100% of the time. Next, an agent coded by ChatGPT, called *gpt_agent* represents a greedy one-step-lookahead player does beat the random agent, but it should be beaten by most of the search algorithms we've discussed in class. Finally, a tournament against the strongest set of agents within the class will be run to determine the rankings. You do not have to win the tournament to get a satisfactory grade, but the winning team(s) will receive 100% of the performance grade.

In all phases, due to the deterministic nature of the game board, each match will consist of N games, with N an even integer, giving both programs equal opportunity to play first. The evaluation phase will require a lot of matches so please be mindful of your program's runtime. We will perform a screening process by pairing your agent with a random agent to ensure the run-times matches the expectations (check the Tournament Constraints in Section 6.1.1).

6.1.1 Tournament Constraints

During the tournament, we will use the following additional rules:

- **Execution Environment**. We will run the tournament on SOCS teaching Linux environment (as in the mimi.cs.mcgill.ca server and desktops in Trottier 3rd floor). These run Linux, Python Python 3.10. We will only install libraries as defined in `requirements.txt`, so you are not allowed to use external libraries. This means built-in libraries for computation are allowed but libraries made specifically for machine learning or AI are not. If you think you would require some external libraries not specified in `requirements.txt` and which is not a AI/ML specific library, please post a question in Ed to get an approval from the TAs, but we will almost always disallow things unless you really convince us we missed something needed.

- **Turn Timeouts**. During each game, your agent will be given no more than 2 seconds to choose its move. Please note that in past years, we gave extra time for the first move, but it was found that no strong players used this time, and it makes marking much longer, so now the rule is 2 seconds each and every move. Use the code provided to see your agent's current timings, and consider using the time library already imported into student_agent for you to terminate your search on time. If your agent exceeds the time limit drastically you will lose some fraction (up to all for infinite loops) of your performance score.

- **Illegal moves**. In the game, if your agent attempts to move to positions which are illegal, i.e. a square that isn't empty or one where no opponent's rocks get flipped,

then the game code makes a random move for you. If your code fails during execution, then the game will also continue with a random move.

- **Multi-threading**. Your agent must be single threaded, run on only one processor and complete all computation when returning from the step function (as in, you are not allowed to "think" on your opponent's time).
- **File IO**. Your player will not be allowed to read and write files : all file IO is prohibited. In particular, you are not allowed to write files, so your agent will not be able to do any learning from game to game.
- **Memory Usage**. Your agent will run in its own process and will not be allowed to exceed 500 mb of RAM. Exceeding the RAM limits will result in a game loss.

You are free to implement any method of choosing moves as long as your program runs within these constraints and is well documented in both the write-up and the code. Documentation is an important part of software development, so we expect well-commented code. All implementation must be your own.

6.2 Report Grading Scheme

The marks for the write-up will be awarded as follows:

- Executive Summary: 5/50
- Detailed explanation: 15/50
- Quantitative Analysis: 15/50
- Pros/cons of Chosen Approach (combined with description of other methods tried, if present): 5/50
- Future Improvements: 5/50
- Organization: 5/50

7. Agent Tips and Heuristics

Of the algorithms we saw in class, those at the end of the games section are most often *strong* agents in the tournament: alpha-beta and MCTS usually alternate as choices of the top team. In 2022 the winning method was titled "Alpha-Beta Pruning with Move Order and Don't Lose Evaluation Heuristic and Game-State Memoization".

However, as your time in the end of term is precious, know that some teams placed in the top half of scores without any large search, simply by coding particularly clever heuristics with greedy approaches. All students we've looked at in the top 20% use methods based on efficient tree search.

Ideas for heuristics to get you started are contained in the greedy corners agent generated by ChatGPT. There are also many tutorials for human Reversi players, and those can certainly be mined for ideas (please cite if used).

8. Academic Integrity

This is a group project. The exchange of ideas regarding the game is encouraged, but sharing of code and reports between different groups or paying someone for a solution is forbidden and will be treated as cheating. We will be using document and code comparison tools to verify that you were truly in charge of creating your own submission.

Please see the syllabus and www.mcgill.ca/integrity for more information.

9. Contact

Please post on Ed with the Projects label with any questions.