COMP 424 Final Project Game: ***Colosseum Survival!***

Course Instructor: David Meger
Due Date: Friday Dec 1st, 2023, 8:59PM EST
Source for this file: `https://www.overleaf.com/read/vnygbjryrxrt#7b70cb`

This document describes the game 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/Project-COMP424-2023-Fall`.

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 ***Colosseum Survival!***

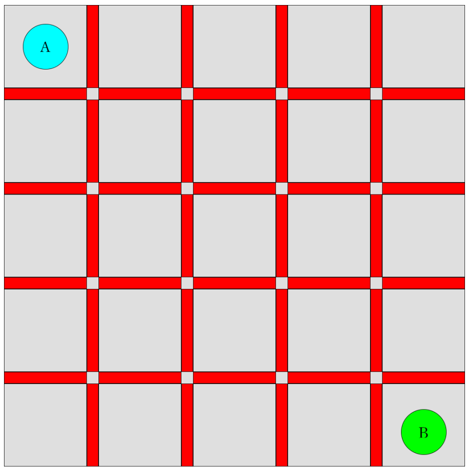


Figure 1: Gameboard

2. Game Description and Rules

Colosseum Survival! is a 2-player turn-based strategy game in which two players move in an $M \times M$ chessboard and put barriers around them until they are separated in two closed zones. M can in principle have any value greater than 1, but you will be evaluated on games from 6 to 12 inclusive. Each player will try to maximize the number of blocks in its zone to win the game.

2.1 Setup

At first, players A and B are randomly positioned on the chessboard symmetrically. The two players will take turns to move in the chessboard and put barriers. In each turn, one player will move **at most** K steps and **must** put a barrier in one of the 4 directions around itself at the end of moving. Here, K is computed as $\lfloor \frac{M+1}{2} \rfloor$, which defines the maximum number of allowable steps. One player cannot go into the other's position, go through the barriers, or put barriers in places that already have barriers (including the game borders). To increase the randomness of the game, less than $K * 2$ barriers will be initially put on the chessboard symmetrically. Each step can only be made horizontally or vertically. That is, moving diagonally requires 2 moves (e.g. first left, then up), and that both the walls along that path are missing.

2.2 Objective

The game ends when two players are separated into two closed zones by barriers and borders. The final score of each player is *the number of blocks in its zone when the game ends*. The player with the higher score will win the game, and will be awarded 1 point. If there is a tie, both players will be awarded 0.5 points. An example gameplay is shown in Figure 2.

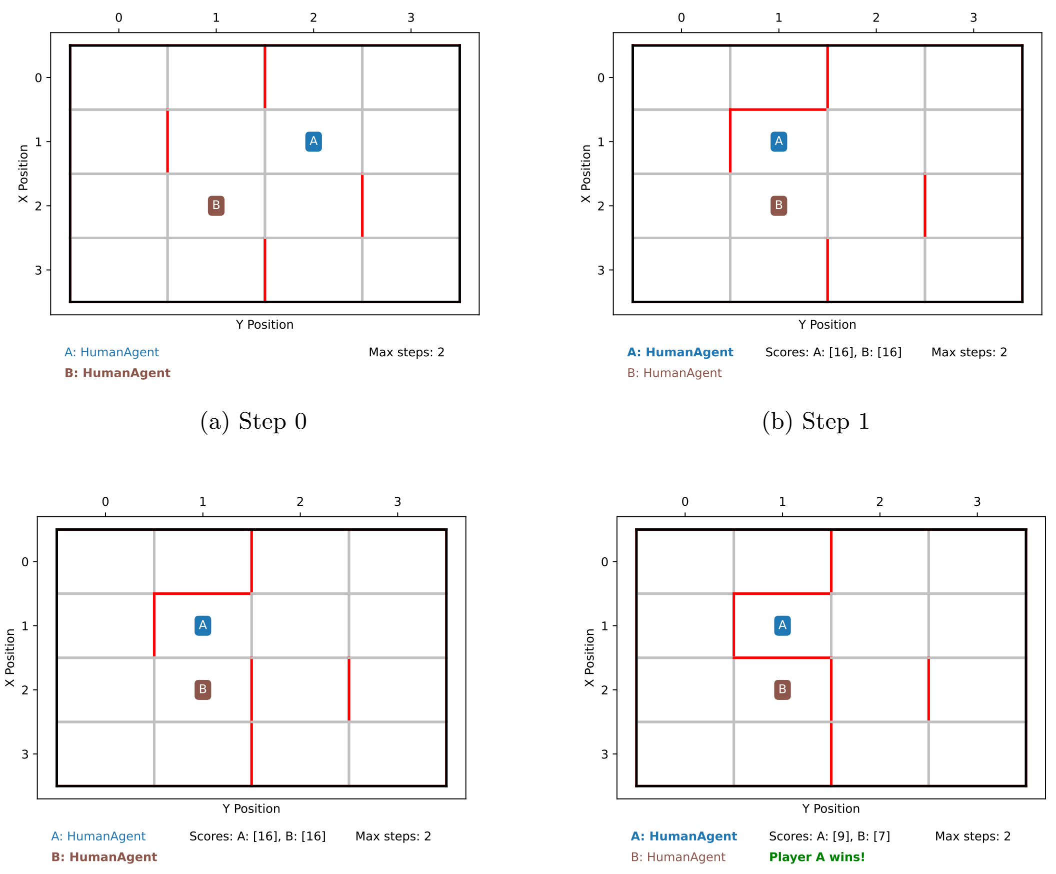
2.3 Playing the game

For each step, the move consists of providing an (x,y) co-ordinate of the board where the player wants to go, and a direction (up/right/down/left) the player wants to put a wall. The number of steps taken to go to position (x,y) will be automatically computed by the game engine using a breadth-first-search algorithm, and if the steps are more than K then errors will be shown.

3. Evaluation

We will use several phases to group your agents by strength. A first pass is that you should definitely be able to beat the random agent provided. Next, make sure you can do well against a human player who has not practiced too much. 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, 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 runtimes matches the expectations (check the Tournament Constraints in Section 7.1.1).



(a) Step 0 (b) Step 1 (c) Step 2 (d) Step 3

Figure 2: A sample game between two agents (A and B) on a 4x4 chessboard, thus having max allowable steps $K = 2$. The board is initialized (Step 0) with random (symmetrical) wall placements. A then moves to position $(1, 1)$ and places a wall on the *top* (Step 1). B remains in the same position $(2, 1)$ and places a wall on the *right* (Step 2). A then wins the game by staying in the same position $(1, 1)$ and placing a wall on *down*, thus having control of 9 blocks in the board (Step 3).

4. Assignment Details

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

4.1 Pre-requirements

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

4.2 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 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. Any helper 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.

5. 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 should use the source of this document [1] as a template to write your report in LaTeX. The report must include the following required components:

- A brief overall motivation or executive summary for your approach. Skip details but give the reader and overview of what was most important overall, what play quality you think you achieved and why and how you view the problem overall (roughly half a page).
- 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 in the size 12 board? Is it the same for all branches, or deeper in some cases than others?
    2. What **breadth** does your agent achieve in the size 12 board? 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?

1. You can find the source of this document in Overleaf here: https://www.overleaf.com/read/vnygbjryrxrt7b70cb. You would need to create an Overleaf account. Copy this project and create a new project to write your report. Replace the `author` information with your own group information.

3. How do the above factors (depth and breadth) scale with board size. Attempt to give a rough big-Oh notation and explain.
4. What were the impact of any heuristics, such as evaluation functions and move ordering? Compare the breadth and depth achievable with and without these factors.
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)
- If you tried other approaches during the course of the project, such as failed/weaker algorithmic ideas, or dummy agents to test against, summarize them briefly and discuss how they compared to your final approach. (half page)
- A brief description of how you would go about improving your player (e.g. by introducing other AI techniques, changing internal representation etc.) (half page)

6. Submission

First, fill your and your team members 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 due to McGill's policy about no assessments during study day and final exams.

7. Grading Scheme

40% of the project grade will be allotted for performance in the tournament, and the other 60% will be based on your report.

7.1 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.

7.1.1 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. which requires more number of steps than K, then we will run a random walk over the game board. 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.

7.2 Report Grading Scheme

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

- Overall motivation: 5/60

- Detailed explanation: 25/60
- Quantitative Analysis: 10/60
- Pros/cons of Chosen Approach (combined with description of other methods tried, if present): 10/60
- Future Improvements: 5/60
- Organization: 5/60

8. 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 20% of scores without any large search, simply by coding strong heuristics with BFS, A^* or similar "simple" methods.

Ideas for heuristics to get you started:

1. Check for being surrounded by 3 walls after you take your turn. This can be an immediate loss if your opponent builds the 4th wall to box you in.
2. Check for how many move options you would have after ending (extension of the above).
3. Check your distance from opponent. Near can be good to trap them (aggressive play), while staying far can be better for survival (defense). Some balance can be good.
4. The overall number of walls in the board is an indicator of what game stage you're at. May play differently during early, middle and end game.
5. In many cases, "continuing" a wall or pattern is more effective than moving to purely free space. Can be a way to prioritize moves.

9. Academic Integrity

This is a group project. The exchange of ideas regarding the game is encouraged, but sharing of code and reports is forbidden and will be treated as cheating. We will be using document and code comparison tools to verify that the submitted materials are the work of the authors only.

You may look at reference implementations of algorithms for inspiration, but limit code directly copy/pasted into your file and ensure to cite the source in both the code file and your report PDF. The same rules apply for code generated by any LLM such as ChatGPT - in this case mention both the tool and the prompt you used.

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

10. Contact

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