COMP 424 Final Project Game: *Colosseum Survival!*

Course Instructor: Jackie Cheung and Bogdan Mazoure
Project TAs: Koustuv Sinha (`koustuv.sinha@mail.mcgill.ca`) **and Shuhao Zheng** (`shuhao.zheng@mail.mcgill.ca`)
Due Date: April 8th, 2022, 11:59PM EST

1. Goal

The main goal of the project for this course is to give you a chance to play around with some of the AI algorithms discussed in class, in the context of a fun, large-scale problem. This year we will be working on a game called ***Colosseum Survival!*** (https://github.com/Ctree35/Project-COMP424-2022-Winter). Koustuv Sinha and Shuhao Zheng are in charge of the project, and should be contacted directly about any bugs in the provided code. General questions should be posted in Ed. **This is a group project, with two students per group**.

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 have a value between 4 and 10. Each player will try to maximize the number of blocks in its zone to win the game.

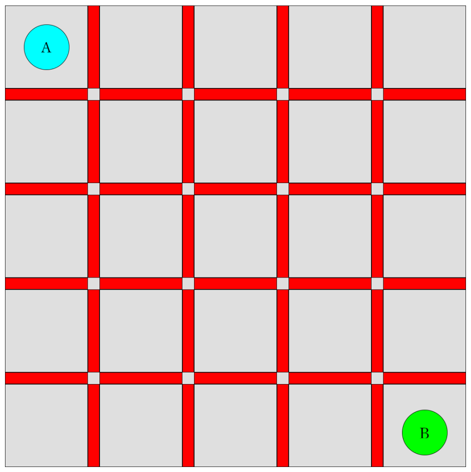


Figure 1: Gameboard

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 $(M+1)/2$, which defines the maximum number of allowable steps. Each step can only be made horizontally or vertically. 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, $K * 2$ barriers will be initially put on the chessboard in a symmetrically.

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 in the endgame.* 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/down/left/right) 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.

2.4 Evaluation

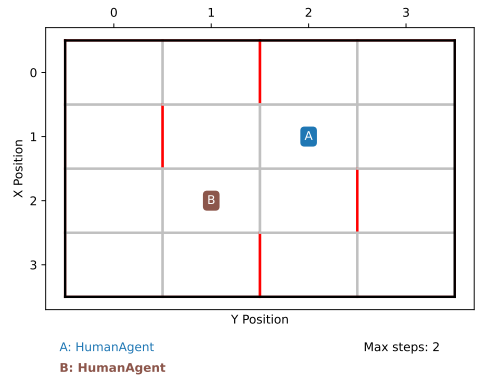
In the first round of the evaluation, we will hold a competition where every submitted program will be playing one match against a random subset of all submissions. Due to the deterministic nature of the game board, each match will consist of N games, giving both programs equal opportunity to play first (typically $N > 1000$). For the second round, the 10 percent highest scoring agents from the first game will play against each other in the playoff round where each agent is paired against all other agents to find the highest scoring agents. 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 5.1.1).

3. Assignment Details

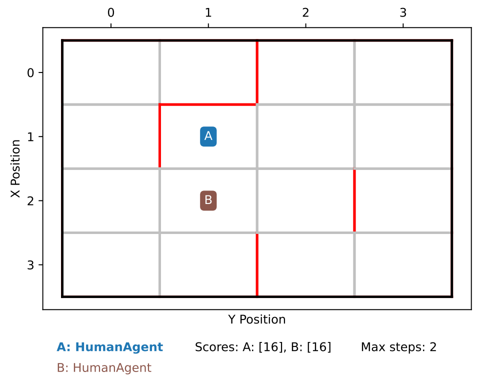
In this final project, your task is to develop an *agent* to play the *Colosseum Survival* game. We have developed a minimalistic game engine in Python, which you will extend to add your own *agents.*

3.1 Pre-requirements

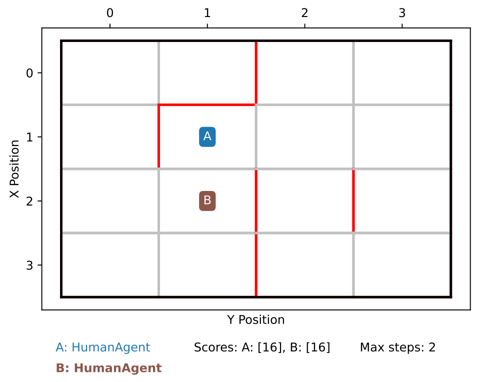
This project you will need to implement agent that playes Colosseum Survival using Python. Specifically, we strongly recommend to brush up Python 3 fundamentals before writing your code. We would also require you to know `git` fundamentals to work with the version control system, and have an account in Github.



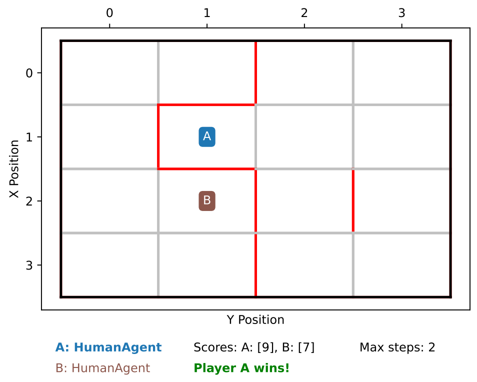
(a) Step 0



(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).

3.2 Working with Github Classrooms

For the purposes of this project, we will be using Github Classrooms. Github Classrooms allow us to easily share the template repository and create private repositories for students. To start working in your assignment:

- Accept this invitation (**https://classroom.github.com/a/dRpR9fvL**) to be added to our COMP 424 Github Classroom. Please register your team by **March 11th, Friday**.
- Choose or create your team. Teams can be formed only with **two** members.

- Once you accept the assignment, your code repository will be setup using the starter code.
- In your repository, a Pull Request (PR) would be automatically created, which would serve as a way to communicate with the TAs about specific, private concerns regarding your code or to get feedback. Alternatively, you can contact the TAs involved in this project directly.
- Clone your assigned repository to start working on the assignment [1].
- Navigate to the cloned directory and install the dependencies : `pip install -r requirements.txt`
- To test that everything runs on your machine as required, run the following command in the cloned directory: `pytest`

3.3 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. Follow the steps to implement and test your own agent:

- Modify the `student_agent.py` file in `agents/` directory, which extends the `agents.Agent` (in `agents/agent.py`) class.
- Implement the step function with your game logic
- Register your agent using the decorator `register_agent`. The `StudentAgent` class is already decorated with the name *"student_agent"*, feel free to change it or keep it the same.
- Import your agent in the `__init__.py` file in `agents/` directory. This step is already done for `StudentAgent`.
- Run and test your agent using `simulator.py` script.

Important: You should not modify any other files apart from `student_agent.py` and any helper file you need should also be created within `agents/` directory. In the event of any update to the game code, the TAs will announce the necessary steps in Ed to pull the starter code.

3.4 Play with your agent

You can use the `simulator.py` script to run an interactive session with your agent! A Human agent (yourself) is defined in `agents/` directory, along with a random walk agent. You can first *visualize* the game happening between the two agents as follows:

1. Check the detailed description of the files in the repository in Section 8

```
python simulator.py \
  --player_1 human_agent \
  --player_2 random_agent \
  --display
```

You can play with your agent likewise by using the name you *registered* your agent with (e.g. *"student_agent"*). To quickly play the game without displaying the UI, remove the `--display` flag.

3.5 Testing and Autoplay

Since the game setup is non-deterministic, your submitted agent will be evaluated on multiple runs by playing them against the `random_agent` or other student submissions. This is achieved by the autoplay mode, enabled by `--autoplay` flag in `simulator.py`. Thus, it is crucial that you test your agent against the `random_agent` before submitting your work. You can play with the flag `--autoplay_runs` to set the number of simulations. In autoplay mode, the starting agent is swapped every other run, so as to remove the disadvantage of starting first in the game.

During autoplay, if your agent does not adhere to the boundary conditions of the board, or attempts to make an invalid move (more than K steps), the game engine will automatically run a *random walk* on behalf of your agent. This may adversely affect your agents performance during evaluation. You should test your agent thoroughly for these edge cases.

3.6 Submission

First, fill your and your team members details in `authors.yaml` file in the repository. You should periodically `commit` your code and push it to the repository assigned to you. Your code will be automatically deemed submitted post the deadline date. You can continue to make changes in the Github Classroom repository before the deadline. Any changes post the deadline will not be used for the final evaluation.

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 should use the source of this document [2] as a template to write your report in LaTeX. The report must include the following required components:

- An explanation of how your program works, and a motivation for your approach.
- A brief description of the theoretical basis of the approach (about a half-page in most cases); references to the text of other documents, such as the textbook, are appropriate

2. You can find the source of this document in Overleaf here: https://www.overleaf.com/read/gcpfjdpqpytp. 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.

but not absolutely necessary. If you use algorithms from other sources, briefly describe the algorithm and be sure to cite your source.

- A summary of the advantages and disadvantages of your approach, expected failure modes, or weaknesses of your program.
- If you tried other approaches during the course of the project, summarize them briefly and discuss how they compared to your final approach.
- A brief description (max. half page) of how you would go about improving your player (e.g. by introducing other AI techniques, changing internal representation etc.)

4.1 Submission

To submit your report, you should also add the report PDF directly in the repository in the `report` folder, rename it to `report.pdf`, and commit and push your report alongside with your code.

5. 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.

5.1 Tournament Grading Scheme

The top scoring agent will receive full marks for the tournament. The remaining agents 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 tournament portion, your agent must beat the random player.

5.1.1 Tournament Constraints

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

- **Execution Environment**. We will run the tournament on Linux environment (CentOS 7), Python 3.9 and install libraries as defined in `requirements.txt`. However, 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.
- **Turn Timeouts**. During each game, your agent will be given no more than 30 seconds to choose its First move, and no more than 2 seconds to choose each subsequent move. The initial 30 second period should be used to perform any setup required by your agent (e.g. loading data from files). If your player does not choose a move within the allotted time, a random move will be chosen instead. If your agent exceeds the time limit drastically (for example, if it gets stuck in an infinite loop) then you will suffer an automatic game loss.

- **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 will be allowed to use multiple threads. However, your agent will be confined to a single processor, so the threads will not run in parallel. Also, you are required to halt your threads at the end of your turn (so you cannot be computing while your opponent is choosing their move).
- **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.

5.2 Report Grading Scheme

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

- Technical Approach: 20/50
- Motivation for Technical Approach: 10/50
- Pros/cons of Chosen Approach: 5/50
- Future Improvements: 5/50
- Language and Writing: 5/50
- Organization: 5/50

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

7. Contact

Feel free to contact the TA's if you have any specific concern that needs to be addressed.

- Koustuv Sinha, koustuv.sinha@mail.mcgill.ca
- Shuhao Zheng, shuhao.zheng@mail.mcgill.ca

8. Detailed description of the repository

The starter code can be viewable here: https://github.com/Ctree35/ Project-COMP424-2022-Winter. The following describes the usage of each component of this code:

```
|-store.py              # Storing agents for decorator
|-world.py              # Main Game engine code
|-requirements.txt      # Specify external libraries to be installed used pip
|-test                  # Unit test files
| |-conftest.py
| |-test_agent.py       # Test case for AI agents
| |-test_world.py       # Test case for game engine
|-ui.py                 # UI Engine files
|-authors.yaml          # Students should fill out author information
|-constants.py          # Handy constants for the game
|-agents                # Directory containing all agents
| |-random_agent.py     # Agent that takes random walk
| |-__init__.py         # All agents should be imported here
| |-human_agent.py      # Human agent using which you can play interactively
| |-agent.py            # Base agent class
| |-student_agent.py    # Implement your AI here
|-README.md
|-utils.py              # Some utilities for the game
|-simulator.py          # Main entry point for running the game
|-report                # Upload your report in this directory
| |-README.md
```