Due Dates	☐ D1: Saturday October 1, 2023 (11:59pm) ☐ D2: Saturday October 22, 2023 (11:59pm) ☐ D3: Saturday December 3, 2023 (11:59pm) Late submissions will be penalized by 10% per day
Teams	You can do the project in teams of at most 3. Teams must submit only 1 copy of the project via the team leader's account. Cross-section teams are allowed.
Purpose	The purpose of this project is to make you implement minimax, alpha-beta and develop adversarial heuristics.

1 AI Wargame

AI Wargame[1] is a 2-player game played by an *attacker* (`a`) and a *defender* (`d`) on a 5×5 board. A demo of the game is available at https://users.encs.concordia.ca/~kosseim/COMP472Fall2023/

Each player has 6 units on the board. These units can be of different types:

AI (`A`) each player has only 1 AI unit. The goal of the game is to destroy the opponent's AI.
Viruses (`V`) are very offensive units; they can destroy the AI in 1 attack.
Techs (`T`) are very defensive units, Tech and Virus are equal in combat against each other.
Programs (`P`) are generic soldiers.
Firewalls (`F`) are strong at absorbing attacks, but weak at damaging other units.

Each unit has an associated health level, represented by an integer between $[0 \ldots 9]$. Initially, all units have full health (9). When the health level of a Virus, a Tech, a Program or a Firewall reaches 0 or below 0, the unit is destroyed and removed from the board. If the health level of an AI reaches 0, then the player loses the game.

1.1 Initial Configuration

At the beginning of a game, the attacker has 1×AI, 2×Viruses, 2× Programs and 1×Firewall; while the defender has 1×AI, 2×Techs, 2×Firewalls and 1×Program. The game starts with the following initial configuration:

```
     0   1   2   3   4
A:  dA9 dT9 dF9  .   .
B:  dT9 dP9  .   .   .
C:  dF9  .   .   .  aP9
D:   .   .   .  aF9 aV9
E:   .   .  aP9 aV9 aA9
```

`dA9`, at `A0`, represents the defender (`d`)'s AI (`A`) and has a health of 9.
`aP9`, at `C4`, represents the attacker (`a`)'s Program (`P`) and has a health of 9.
...

The attacker starts the game.

[1] Do not bother googling this game, I invented it.

S \ T	AI	Virus	Tech	Fire-wall	Pro-gram
AI	3	3	3	1	3
Virus	9	1	6	1	6
Tech	1	6	1	1	1
Firewall	1	1	1	1	1
Program	3	3	3	1	3

Table 1: Damage values (with adversarial units)

S \ T	AI	Virus	Tech	Fire-wall	Pro-gram
AI	0	1	1	0	0
Virus	0	0	0	0	0
Tech	3	0	0	3	3
Firewall	0	0	0	0	0
Program	0	0	0	0	0

Table 2: Repair values (with friendly units)

1.2 Actions

Each player take turn to play any of the following actions.

Movement A single unit can move to an adjacent position on the board. Rules to move a unit are the following:

1. The destination must be free (no other unit is on it).
2. Units are said to be *engaged in combat* if an adversarial unit is adjacent (in any of the 4 directions). If an AI, a Firewall or a Program is engaged in combat, they cannot move.
   The Virus and the Tech can move even if engaged in combat.
3. The attacker's AI, Firewall and Program can only move up or left.
   The Tech and Virus can move left, top, right, bottom.
4. The defender's AI, Firewall and Program can only move down or right.
   The Tech and Virus can move left, top, right, bottom.

Attack A unit S can attack another unit T. Rules to attack a unit are the following:

1. T must be adjacent to S in any of the 4 directions (up, down, left or right).
2. T and S must be adversarial units (i.e. belong to different players).

A combat is bi-directional. This means that if S attacks T, S damages T but T also damages S.
Table 1 shows the damages inflicted to an adversary's health during a combat.
Health values cannot go above 9 or below 0. If a unit's health reaches 0, it is destroyed and eliminated from the board.

Repair A unit S can repair another unit T. Rules to repair a unit are the following:

1. T must be adjacent to S in any of the 4 directions (up, down, left or right).
2. T and S must be friendly units (i.e. belong to the same player).
3. The repair must lead to a change of health on T. Table 2 shows the change in health resulting from repairs. So for example,
   - a Tech cannot repair a Virus (see value 0 in Table 2). This is would be an invalid action.
   - S cannot repair T if T's health is already at 9. This is would be an invalid action.

Self-Destruct Any unit can kill itself (and be removed from the board) and damage surrounding units.
Self-destruction removes the unit and inflicts 2 points of damage to all 8 surrounding units (if present). This includes diagonals and friendly units. Remember that health values cannot go above 9 or below 0. If the health of a unit reaches 0 or below 0, it is removed from the board.

1.3 End of the Game

The game ends when a player loses their AI, or a pre-determined number of moves has been reached (e.g. 100). A player wins if their AI is alive while the other AI is destroyed; otherwise the defender wins (because the attacker started playing first).

Given that any unit can self-destruct, no player should be left in a position where no action is available to them.

2 Your Task

In this project, you will implement an adversarial search to play the game of AI Wargame automatically. You will implement a minimax algorithm + alpha-beta pruning and at least 2 heuristics.

2.1 Play Modes

Your program should be able to run in manual mode and in automatic mode. This means that you should be able to run your program with:

1. manual entry for both players (i.e. human vs human: H-H)
2. manual entry for one player, and automatic moves for the other (i.e. human vs AI: H-AI, AI-H)
3. automatic moves for both players (i.e. AI vs AI: AI-AI)

After each move, your program must display the new configuration of the board, and some statistics (see Section 2.5).

2.2 Programming Environment

1. You must use GitHub to develop your project so we can monitor the contribution of each team member (make sure your project is private while developing).
2. To program the project, you can use any of the following languages: Python, Java, C, C++. If you wish to use another language, please check with me first (just send me an email). However, note that a significant skeleton of code written in Python 3.11 is available on Moodle. You can use this code as a starting point. If you wish to use another language, you will have to write on your own all the functionality provided in the Python skeleton.
3. If you do choose Python, if you wish, you can use the `PyPy`[2] runtime instead of the regular `CPython` to make your code run faster, but be careful as `Pypy` is not compatible with all `Python` modules.

2.3 The Heuristics

Develop 2 different heuristics, e_1 and e_2, to play this game automatically. For deliverable D2 (see Section 3), you will need to implement and demo your code with the following heuristic:

$$e_0 = (3V_{P1} + 3T_{P1} + 3F_{P1} + 3P_{P1} + 9999AI_{P1}) - (3V_{P2} + 3T_{P2} + 3F_{P2} + 3P_{P2} + 9999AI_{P2})$$

where:

V_{Pi} = nb of Virus of Player i $\quad$ T_{Pi} = nb of Tech of Player i $\quad$ F_{Pi} = nb of Firewall of Player i
P_{Pi} = nb of Program of Player i $\quad$ AI_{Pi} = nb of AI of Player i

[2] https://www.pypy.org/

2.4 The Input

It is not necessary to have a fancy user-interface. A simple command-line interface with a text-based I/O is sufficient. As long as your program supports the following input:

2.4.1 Game Parameters

Your program must accept the following game parameters[3].

1. The maximum allowed time (in seconds) for your program to return a move
   Your AI should not take more than `t` seconds to return its move. If it does, your AI will automatically lose the game. This entails that even if your adversarial search is allowed to go to a depth of `d` in the game tree, it may not have time to do so every time. Your program must monitor the time, and if not enough time is left to explore all the states at depth `d`, it must interrupt its search at depth `d` and return values for the remaining states quickly before time is up.
2. The maximum number of turns to declare the end of the game (see Section 1.3)
3. A Boolean to force the use of either minimax (`FALSE`) or alpha-beta (`TRUE`)
4. The play modes
   Your program should be able to make either player be a human or the AI. This means that you should be able to run your program in all 4 combinations of players: H-H, H-AI, AI-H and AI-AI.

2.4.2 Coordinates of Actions

Moves will be specified by the coordinates of the source S, then the coordinates of the target T (e.g. `B2 B3`).

- If the target T is an empty cell, the action will be interpreted as a movement.
- If the target T is an adversarial unit, the action will be interpreted as an attack.
- If the target T is a friendly unit ($\neq S$), the action will be interpreted as a repair.
- If $T = S$, the action will be interpreted as a self-destruction.

Coordinates will be specified by the row number (a letter $\in [A...E]$), then the column number (an integer $\in [0...4]$); for example, `B3`.

If a human player enters an illegal action (e.g. `W3`, the coordinates of a position that is not empty, a move of a unit engaged in combat ...), then the human will only be warned and be given a chance to enter another move with no penalty[4].

If your AI generates an illegal action it will automatically lose the game.

[3] You do not need to check the validity of these input values; you can assume that they will be valid.

[4] The point of this rule is to avoid losing a game in the tournament (see Section 3.2) because a human did not transcribe your AI's action correctly.

2.5 The Output

Your program should generate an output file with the trace of a single game. The name of these output files should follow the format: `gameTrace-<b>-<t>-<m>.txt`, where `b` is either `false` if alpha-beta is off or `true` if alpha-beta is active; `t` is the value of the timeout in seconds; and `m` is the max number of turns.
For example `gameTrace-true-5-100.txt`, is a game trace where alpha-beta is active, the timeout is 5 seconds for each turn, and the game ends after at most 100 turns.
The trace should contain:

1. The game parameters (see Section 2.4.1):
   a) the value of the timeout in seconds `t`
   b) the max number of turns
   c) if a player is an AI, whether alpha-beta is `on` or `off`
   d) the play modes (eg. player 1 = AI & player 2 = H)
   e) in addition, if a player is an AI, indicate the name of your heuristic: `e0`, `e1` or `e2`
2. The initial configuration of the board.
3. Then, for each action, display:
   3.1. information about the action:
      a) the turn number (eg. `turn #1`)
      b) the name of the player (eg. `Attacker`)
      c) the action taken (eg. `move from C4 to C3`)
      d) if a player is an AI, the time for the action in seconds (eg. `time for this action:  0.4 sec`)
      e) if a player is an AI, the heuristic score of the resulting board: (eg. `heuristic score:  -881`)
      f) the new configuration of the board
   3.2. if a player is an AI, cumulative information about the game so far:
      a) the number of states evaluated by the heuristic function since the beginning of the game (e.g. `Cumulative evals:  18.0M`)
      b) the same number as above but split over each depth (consider the starting node to be at depth 0) (e.g. `Cumulative evals by depth:  1=39 2=277 3=2.3k 4=17.9k 5=95.4k 6=6.5M 7=11.4M`)
      c) the same number as above but expressed as percentages (e.g. `Cumulative % evals by depth:  5=0.5% 6=36% 7=63%`)
      d) the average branching factor: (e.g. `Average branching factor:  4.8`)
4. The winner of the game (e.g. `Defender wins in 37 turns`)

3 Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to project.
The team grade will count for 90% and the individual grade will count for 10%.

Deliverable	Deadline	Functionality	%	Evaluation Criteria
D1 + Demo 1	see page 1	manual game: H-H (no minimax, no heuristic)	30%	Code (25%): functional criteria (I/O, functionality of the rules of the game, detection of illegal actions, output files ...), programming quality. Demo (5%): quality of the presentation, knowledge of the code and clarity of explanations, time management, team QA
D2 + Demo 2	see page 1	same as above + AI moves (minimax, alpha-beta, e_0, e_1 and e_2)	45%	Code (40%): functionality of minimax & alpha-beta (generation and heuristic evaluation of states, return of the best action, ...), quality of the heuristics, programming quality. Demo (5%): quality of the presentation, time management, team QA
Tournament	tbd	same as above	5%	Participation at the tournament with a functional code (i.e. no crashing or other functional error). Your ranking at the tournament will not be evaluated, only your participation.
D3	see page 1	same as above + Report	10%	Clarity of the report, justification of the heuristic, depth of the analysis, presentation, grammar, ...
Individual grade			10%	Peer-evaluation done after D1 and D3 Contribution of each student as indicated on GitHub Individual Q/A at demo 1 and demo 2 (correct and clear answers to questions, knowledge of the program, ...).
		Total	100%	

3.1 Demos

You will have to demo your project after each deliverable. Regardless of the demo time, you will demo the program that was uploaded by the submission deadline. The schedule of the demos will be posted on Moodle. The demos will consist of 2 parts: a presentation and a Q/A part.

Demo 1

- ☐ Prepare a ≈ 5 minute presentation to explain your code and show its functionality.
- ☐ After your presentation, your TA will proceed with a ≈ 10 minute question period. Each student will be asked questions on the code/project, and they will be expected to understand and explain any part of the code. Hence every member of team is expected to attend the demo.
- ☐ At the end of the demo, your TA may ask you to generate output files (see Section 2.5) for specific game parameters (see Section 2.4.1). You will need to generate these files, and submit them on EAS.

Demo 2

- ☐ Prepare a ≈ 5 minute presentation to explain your code and your heuristics e_1 and e_2. Discuss the effect of alpha-beta pruning and the quality of the heuristics. Show actual data to back up your claims.
- ☐ After your presentation, your TA will proceed with a ≈ 10 minute question period. Each student will be asked questions on the code/project, and they will be expected to understand and explain any part of the code and any part of the analysis. Hence every member of team is expected to attend the demo.
- ☐ At the end of the demo, your TA may ask you to generate output files (see Section 2.5) for specific game parameters (see Section 2.4.1). You will need to generate these files, and submit them on EAS.

3.2 Tournament

We will organize an AI versus AI tournament. If you participate and your program does not crash or produces a functional error, then you get 5%. Your ranking at the tournament will not be evaluated, only your participation. More details on the tournament will be posted on Moodle.

3.3 Report

The last deliverable will be a 4-5 page report that describes your heuristics, analyses them with respect to one another and within the scope of the tournament, the effect of alpha-beta pruning on the efficiency and quality of the search etc Show actual data to back up your claims. Also provide an analysis of what you have learned, and a description of what you did right/wrong and what you would change if you were to redo the project today.

4 Submission

4.1 Team Leader

If you work in a team, identify one member as the team leader and make sure this information is indicated in the Groups on the Moodle page. The only additional responsibility of the team leader is to upload all required files (including the files at the demo) from their account and book the demo on the Moodle scheduler.

4.2 Submission Checklist

D1:

- ☐ Create one zip file containing your code, and a `README` file.
- ☐ Name your zip file: `472_Project_ID1_ID2_ID3.zip` where `ID1` is the ID of the team leader.
- ☐ Have the team leader upload the zip file on EAS as Programming Project Group **1** / Submission Number **1**.

During your actual demo with the TA:

- ☐ Generate the output files for the data that the TA will give you.
- ☐ Create a zip file called: `472_Demo1_ID1_ID2_ID3` where `ID1` is the ID of the team leader.
- ☐ Have the team leader upload the zip file on EAS as Programming Project Group **1** / Submission Number **2**.

D2: same as D1, but have the team leader upload:

- ☐ Upload the zip file on EAS as Programming Assignment Group **2** / Submission Number **1**.
- ☐ Upload the demo output zip file on EAS as Programming Project Group **2** / Submission Number **2**.

D3:

- ☐ Save your report in PDF format.
- ☐ Name your report: `472_Report_ID1_ID2_ID3.pdf` where `ID1` is the ID of the team leader.
- ☐ Have the team leader upload the zip file on EAS as Programming Project Group **3** / Submission Number **1**.

Have fun!