Due date November 8, 2021
Late Submission 30% per day
Teams You can do the mini-project in teams of at most 3.
Teams must submit only 1 copy of the project via the team leader's account.
Purpose In this project, you will implement the 2-player game of a *Line 'em Up*.

1 Game of Line 'em Up

The game of *Line 'em Up* is a generic version of *tic-tac-toe*: it is an adversarial 2 player game played on a `n`×`n` board where each position has one of 4 values:

- a white piece (○) – for the player who plays the white pieces
- a black piece (●) – for the player who plays the black pieces
- a bloc (☒) – where no player can place their pieces (in *tic-tac-toe*, there are no blocs)
- an empty position (☐) – where players can place their pieces

Game set up: Initially, `b` blocs (☒) are placed at specific positions, and the rest of the positions are left empty. Each player chooses the color they will play (○ or ●). The player who plays ○ always plays first. Figure 1 shows an initial game board with `n=5` and `b=6` blocs are positioned at `[(0,D), (1,B), (2,A), (2,D), (3,B), (4,C)]`.

	A	B	C	D	E
0				☒	
1		☒			
2	☒			☒	
3		☒			
4			☒		

Figure 1: Example of an initial board with `n=5` and `b=6` blocs at positions `[(0,D), (1,B), (2,A), (2,D), (3,B), (4,C)]`.

Moves: Players play in alternating turns. At each turn, the active player chooses one empty position where they place their own color (○ or ●). Then the next player makes their move.

End game: The game ends as soon as one player has lined up `s` consecutive pieces of their own color in a row, in a column or diagonally; in that case, that player wins the game. If there are no more empty positions on the board, then the game ends in a draw. For example, with the game board of Figure 1, possible wins are shown in Figures 2 and 3 if `s=4`, and a possible draw is shown in Figure 4.

	A	B	C	D	E
0	○	○		☒	
1	●	☒			
2	☒	●		☒	
3	○	☒	●		
4		○	☒	●	

Figure 2: Example of a win for ● with `n=5`, `b=6` and `s=4`.

	A	B	C	D	E
0	○	○	●	☒	●
1	●	☒	●	●	○
2	☒	●		☒	○
3	○	☒	●	●	○
4	●	○	☒	○	○

Figure 3: Example of a win for ○ with `n=5`, `b=6` and `s=4`.

	A	B	C	D	E
0	○	○	○	☒	●
1	●	☒	●	●	●
2	☒	○	○	☒	○
3	○	☒	○	○	●
4	●	○	☒	●	●

Figure 4: Example of a draw with `n=5`, `b=6` and `s=4`.

2 Your Task

Implement an adversarial search for the game of *Line 'em up* for values of `n`, `s` and `b` given as input.

2.1 Skeleton Code

To get you started, a skeleton code which implements a standard *tic-tac-toe* game (i.e. with `n=3`, `b=0` and `s=3`) is available on Moodle. This skeleton includes:

1. an already coded mini-max function,
2. an already coded alpha-beta function, and
3. a basic interface.

The code evaluates the states at the leaves of the game-tree, hence it is not really usable for values of `n>3` in a reasonable time. You must expand this code to:

1. account for the specificity of the game *Line 'em Up* described in Section 1,
2. develop 2 heuristics `e1` and `e2` – see Section 2.3,
3. accept as input a number of parameters – see Section 2.4, and
4. output a variety of information – see Section 4.1.

2.2 Programming Environment

To program the project, you must use Python 3.7. Using the `PyPy`[1] runtime instead of the regular `CPython` will make your code run significantly faster, but be careful as `Pypy` is not compatible with all `Python` modules. You must also use GitHub to develop your project so we can monitor the contribution of each team member (make sure your project is private while developing).

2.3 The Heuristics

Develop 2 different heuristics (`e1` and `e2`) for this game. I recommend to have `e1` be a very simple but fast heuristic, and have `e2` be more sophisticated and complex to compute. This way, you can better appreciate the trade-off between the execution time and the quality of the estimate in order to win the game given reasonable time constraints.

[1] 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:

1. **Game Parameters**: Your program must accept the following game parameters[2].
   (a) the size of the board – `n` – an integer in `[3..10]`
   (b) the number of blocs – `b` – an integer in `[0..2n]`
   (c) the positions of the blocs – `b` board coordinates
   (d) the winning line-up size – `s` – an integer in `[3..n]`
   (e) the maximum depth of the adversarial search for player 1 and for player 2 – 2 integers `d1` and `d2`
   (f) the maximum allowed time (in seconds) for your program to return a move – `t`
   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.
   (g) a Boolean to force the use of either minimax (`FALSE`) or alphabeta (`TRUE`) – `a`
   (h) 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. **Coordinates of Moves**: If at least one player is a human, then their moves will be specified by first indicating the column (numbered `[A..J]`), then the row (numbered `[0..(n-1)]`), for example `B 3`. Note that if a human player enters an illegal move (e.g. `W 3` or the coordinates of a position that is not empty), then they will only be warned and be given a chance to enter another move with no penalty[3]. However, if your program generates an illegal move, then it will automatically lose the game.

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

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

2.5 The Output

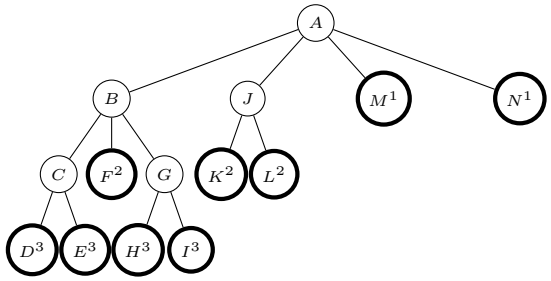
Your program should generate 2 types of output files: game traces and a scoreboard with statistics of multiple games. Sample output files are available on Moodle. Download them now, so you can better follow the description below.

2.5.1 Game Trace Files

Game traces are meant to show the evolution of a single game where both players are controlled by an AI. The name of these output files should follow the format: `gameTrace-<n><b><s><t>.txt` (for example `gameTrace-5642.txt`). Each trace file should contain:

1. The parameters of the game: the values of `n`, `b`, `s`, `t`
2. The position of the blocs
3. The parameters of each player: the values `human` or `AI`, and in the case of an `AI`, the corresponding values of the depth of the search (`d1` or `d2`), the corresponding values of the alphabeta vs minimax of the search (`a1` or `a2`, or both) and which heuristic was used (`e1` or `e2`).
4. A display of the initial configuration of the board.
5. Then, for each move, display:
   (a) the move taken (eg. `B 4`)
   (b) the new configuration of the board
   (c) the following statistics:
      i. The evaluation time of the heuristic (in seconds)
      ii. The number of states evaluated by the heuristic function
      Your code must count and return the number of states that your heuristic function evaluated to select its current move.
      iii. The number of states evaluated at each depth (consider the root to be at depth 0)
      The number of states evaluated at each depth. Here you display, how many nodes were evaluated at depth 1, at depth 2, ...
      iv. The average depth (AD) of the heuristic evaluation in the tree
      As explained above, your program must decide on the next move within `t` seconds; hence a search of all nodes at level `d` may not be feasible. Here, you must compute the average depth of all states evaluated by your heuristic function.
      v. The average recursion depth (ARD) at the current state
      see the example below for an explanation.

   The following example illustrates these statistics. Assume that your search has run the heuristic function on the nodes in bold of the game tree below. The depth of these nodes are indicated as a superscript next to the node name. The statistics to display are then:



i. the total time your heuristic took to run on all 9 nodes $[D, E, F, H, I, K, L, M, N]$

ii. 9

iii. [depth1=2, depth2=3, depth3=4]

iv. $\frac{(3+3+2+3+3+2+2+1+1)}{9} = 2.22$

v. the ARD at node A is calculated recursively, this way:

$ARD(C) = \frac{d(D)+d(E)}{2} = \frac{3+3}{2} = 3$

$ARD(G) = \frac{d(H)+d(I)}{2} = \frac{3+3}{2} = 3$

$ARD(B) = \frac{ARD(C)+d(F)+ARD(G)}{3} = \frac{3+2+3}{3} = 2.67$

$ARD(J) = \frac{d(K)+d(L)}{2} = \frac{2+2}{2} = 2$

$ARD(A) = \frac{ARD(B)+ARD(J)+d(M)+d(N)}{2} = \frac{2.67+2+1+1}{4} = 1.67$

6. Then, at the end of the game, display:
   (a) the winner
   (b) for each heuristic, the following statistics:
      i. The average evaluation time of the heuristic for each state evaluated (in seconds)
      ii. The number of states evaluated by the heuristic function during the entire game
      iii. The average of the per-move average depth of the heuristic evaluation in the tree – i.e. the average of statistic 4(c)iii above for all moves
      iv. The total number of states evaluated at each depth during the entire game
      v. The average of the per-move average recursion depth - i.e. the average of statistic 4(c)v above for all moves
      vi. The total number of moves in the game

2.5.2 Scoreboard File

You will run a series of 2×`r` games (for example 2×10), in AI vs AI mode, where each player uses a different heuristic (eg. player 1 uses `e1` and player 2 uses `e2`), and each player plays as ∘ `r` times, and plays as • `r` times. You will store in an output file called `scoreboard.txt`, the following information:

1. The parameters of the game (the values of `n`, `b`, `l`, `t`)
2. The parameters of each player: their max search depth (`d1` and `d2`)), their alphabeta or minimax (`a1` and `a1`) and their heuristic
3. The number of games played (the value of 2×`r`)
4. The number and percentage of wins for heuristic `e1` and for heuristic `e2`
5. All the information displayed at the end of a game (see Section 2.5.1), but averaged over 2×`s` games

For each series of games, append the statistics of the series at the end of the `scoreboard.txt` file.

2.6 Experiments and Analysis

Once your code is running, generate a sample for one game trace for the following configurations:

1. `gameTrace-4435.txt` (i.e: `n=4`, `b=4`, `s=3`, `t=5`)
   with `d1=6`, `d2=6`, `a1=FALSE`, `a2=FALSE` with the blocs at `[(0,0),(0,4),(4,0),(4,4)]`
2. `gameTrace-4431.txt`: with `d1=6`, `d2=6`, `a1=TRUE`, `a2=TRUE` with the blocs at `[(0,0),(0,4),(4,0),(4,4)]`
3. `gameTrace-5441.txt`: with `d1=2`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions
4. `gameTrace-5445.txt`: with `d1=6`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions
5. `gameTrace-8551.txt`: with `d1=2`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions
6. `gameTrace-8555.txt`: with `d1=2`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions
7. `gameTrace-8651.txt`: with `d1=6`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions
8. `gameTrace-8655.txt`: with `d1=6`, `d2=6`, `a1=TRUE`, `a2=TRUE` with blocs at random positions

and a `scoreboard.txt` for a series of 10 games using the same configurations as above (i.e. 10 games with configuration (1) + 10 games with configuration (2) ...).

Analyse your scoreboard to compare and contrast the effect of the depth, alpha-beta and the two heuristic(s). Prepare a few slides explaining your heuristics and presenting and analysing the above data. You will present these slides at the demo (see Section 4.2).

3 Tournament (Just for fun)

Just for the fun of it, we will organize an AI versus AI tournament. This tournament will not count for points, we are just doing this for fun (and for the thrill of having your team publicly acknowledged on the Moodle page as the winner of the tournament!). For the tournament, we strongly encourage you to use `PyPy`[4]. More details on the tournament will be posted on Moodle.

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

4 Submission

The submission of the mini-project will consist of 3 deliverables:

1. The code & output files
2. The demo (8 min presentation & Q/A)

4.1 The Code & Output files

Submit all files necessary to run your code in addition to a `readme.md` which will contain specific and complete instructions on how to run your experiments. If the instructions in your readme file do not work, are incomplete or a file is missing, you will not be given the benefit of the doubt. Generate the 8 game traces and the scoreboard output files specified in Section 2.6.

4.2 The Demos

You will have to demo your mini-project for $\approx$ 12 minutes. Regardless of the demo time, you will demo the program that was uploaded as the official submission. The schedule of the demos will be posted on Moodle. The demos will consist of 2 parts: a presentation $\approx$ 8 minutes and a Q/A part ($\approx$ 4 minutes). Note that the demos will be done via Zoom and will be recorded.

4.2.1 The Presentation

Prepare an 8-minute presentation to analyse the results of your analysis (see Section 2.6. In particular, discuss the effect of the depth, the use of alpha-beta and the two heuristic(s) functions. Back-up your claims with actual data from your experiments. Prepare a few slides explaining your heuristics and presenting and analysing the above data. You will present these slides at the demo (see Section 4.2). The intended audience of your presentation is your TAs. Hence there is no need to explain the theory. Your presentation should focus on **your** work and analysis.

Any material used for the presentation (slides, ...) must be uploaded on EAS before the due date.

4.2.2 Q/A

After your presentation, your TA will proceed with a $\approx$ 4 minute question period. Each student will be asked questions on the code/mini-project, and he/she will be required to answer the TA satisfactorily. Hence every member of team is expected to attend the demo.

In addition, your TA may give ask you to change your code and ask you to run it again. The output files generated at demo time will have to be uploaded on EAS during your demo.

5 Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to project.

Individual grades will count for 15% and will be based on:

1. a peer-evaluation done after the submission.
2. the contribution of each student as indicated on GitHub.
3. the Q/A of each student during the demo (correct and clear answers to questions, knowledge of the program, ...).

The team grade will count for 85% and will be based on:

Code	functionality, design, programming style, ...	45
Heuristics – $e1$ and $e2$	quality, originality, ...	5
Output files	format, content, correctness	10
Demo presentation	depth of the analysis, clarity and conciseness, presentation, time-management, ...	15
Output files at demo time	format, content, correctness	10
Total		85

5.1 Submission Schedule

Each deliverable is due on the date indicated below.

Deliverable	**Due Date**	**Upload as**
Submit your code, output files, presentation material	November 8, 2021, 11:59pm	Assignment 2
Submit the output files generated at demo time	during your demo	Assignment 20 (yes! 20)

5.2 Submission Checklist

In your GitHub project, include a `README.md` file that contains:

1. on its first line: the URL of your GitHub repository,
2. specific and complete instructions on how to run your program.

Code & Output files

- ☐ Create one zip file containing your code, the output files and the `README.md` file.
- ☐ Name your zip file: `472_Assignment1_ID1_ID2_ID3.zip` where `ID1` is the ID of the team leader.
- ☐ Have the team leader upload the zip file at: https://fis.encs.concordia.ca/eas/ as `Assignment2`.

Demo & Output files During your actual demo with the TA:

- ☐ Prior to your demo, make your GitHub repository public.
- ☐ Generate the output files for the dayaset set 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 at: https://fis.encs.concordia.ca/eas/ as `Assignment20`.

Have fun!