
Due Dates	☐ D1: Thursday February 13, 2025 (11:59pm) ☐ D2: Thursday March 13, 2025 (11:59pm) ☐ D3: Thursday April 3, 2025 (5:45pm) 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.
Skeleton code	A skeleton code is provided on Moodle. You can use this code as a starting point to write the rest of the deliverables.

---

1 Mini Chess

Mini Chess is a 2-player game, inspired by the game of chess, played with the white pieces and the black pieces on a 5×5 board.

Each player (white/black pieces) has 6 pieces on the board. These pieces are:

King (`K`): The King moves 1 square in any direction. The goal of the game is to capture the opponent's king
Queen (`Q`): The Queen moves any number of squares in any direction.
Bishop (`B`): The Bishop moves diagonally.
Knight (`N`): The Knight moves in an L-shape (2 squares in one direction and 1 in a perpendicular direction).
pawns (`p`): Each player has 2 pawns. Pawns move 1 square forward, capturing diagonally. Pawns promote to a Queen upon reaching the 5th row opposite their side.

1.1 Initial Configuration

At the beginning of a game, both the player with the white and black pieces have 1×King, 1×Queen, 1×Bishop, 1×Knight, 2×pawns. The game starts with the following initial configuration:

```
5   bK  bQ  bB  bN   .
4    .   .  bp  bp   .
3    .   .   .   .   .
2    .  wp  wp   .   .
1    .  wN  wB  wQ  wK

     A   B   C   D   E
```

`wK`, at `E1`, represents the (`w`)hite (`K`)ing (`wK`).
`bp`, at `C4`, represents the (`b`)lack (`p`)awn (`bp`).
...

The player with the white pieces starts the game.

1.2 Actions

Each player takes turns to move a single piece. Captures happen when your piece moves over one of the opponent's pieces. In such a situation, the opponent's piece is removed from the board.

The King: The King is the most important piece in the game. It can move a single square in any direction. Unlike regular chess, the king can move to positions where it is under attack and can get captured. Capturing the opponent's king results in a win.

The Queen: The Queen is typically the most powerful piece in a regular game of chess. She is able to move any number of squares in any direction (except in the L shape the Knight moves in).

The Bishop: The Bishop can only move diagonally any number of squares, in any direction.

The Knight: The Knight moves in an L-shape the same way as regular chess (2 squares in one direction and 1 in a perpendicular direction) in any direction. It is the only piece capable of jumping over other pieces. In the example board from page 1, `wN` can move to `A3`, `C3`, or `D2`.

The pawn: The pawn is the least powerful piece in the game, it moves a single square forward and can only capture pieces that are a single square diagonally in front of it. However, when reaching the other side of the board (row 5 for white and row 1 for black), the pawn is promoted to a Queen.

No other chess mechanics are to be implemented such as check, checkmate, en-passant, or pawn promotion to pieces other than Queens. Additionally, pawns cannot move 2 squares on their first move.

1.3 End of the Game

The game can either end in a **win** or a **draw**.

Win: A win occurs when you capture your opponent's king.

Draw: A draw occurs if the number of pieces on the board has not changed in 10 turns. A turn is done when white has made a move and black has made a move.

2 Your Task

In this project, you will implement an adversarial search to play the game of Mini Chess 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 must use Python. There is a skeleton code provided on Moodle written in Python. You can use this code as a starting point.
3. If you wish, you can use the `PyPy`[1] 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:

$$\begin{aligned} e_0 = \quad & (\#wp + 3 \cdot \#wB + 3 \cdot \#wN + 9 \cdot \#wQ + 999 \cdot wK) \\ & -(\#bp + 3 \cdot \#bB + 3 \cdot \#bN + 9 \cdot \#bQ + 999 \cdot bK) \end{aligned}$$

where: $\#wp$ is the number of white pawns, $\#bB$ is the number of black Bishops, ...

[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:

2.4.1 Game Parameters

Your program must accept the following game parameters[2].

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 square, that piece will simply move to the new square.
- If the target T is occupied by one of the opponent's pieces, the opponent's piece will be captured and your piece will move to the new square.
- If the target T is occupied by one of your pieces, this is an illegal move.

Coordinates will be specified by the row number (a letter $\in [A...E]$), then the column number (an integer $\in [1...5]$); for example, `B3`.

If a human player enters an illegal action (e.g. `W3`, the coordinates of a square occupied by one of your pieces, a square not in the board, ...), then the human will only be warned and be given a chance to enter another move with no penalty[3].

If your AI generates an illegal action 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 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) the play modes (eg. player 1 = AI & player 2 = H)
   d) if a player is an AI, whether alpha-beta is `on` or `off`
   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 name of the player (eg. `white`)
      b) the turn number (eg. `turn #1`)
      c) the action taken (eg. `move from C3 to C4`)
      d) if a player is an AI, the time for the action in seconds (eg. `time for this action: 0.05 sec`)
      e) if a player is an AI, the heuristic score of the resulting board: (eg. `heuristic score: -3`)
      f) if a player is an AI, the heuristic score of the resulting minimax/alpha-beta search: (eg. `alpha-beta search score: -9`)
      g) 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 explored since the beginning of the game (e.g. `cumulative states explored: 2.2M`)
      b) the same number as above but split over each depth (consider the starting node to be at depth 0) (e.g. `cumulative states explored by depth: 1=144 2=1.1k 3=7.3k 4=46.7k 5=286.5k 6=1.8M`)
      c) the same number as above but expressed as percentages (e.g. `cumulative % states explored by depth: 3=0.3% 4=2.2% 5=13.3% 6=84.1%`)
      d) the average branching factor: (e.g. `average branching factor: 6.7`)
4. The winner of the game (e.g. `white won in 13 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 Moodle.

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 Moodle.

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 2 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 Moodle.

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 Moodle.

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

- ☐ Upload the zip file on Moodle.
- ☐ Upload the demo output zip file on Moodle.

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 pdf file on Moodle.

Have fun!