LINFO1361: Artificial Intelligence
Assignment 2: Adversarial search (Shobu)

Achille Morenville, Benoît Ronval, Eric Piette

February 2024

Guidelines

- The Alpha-Beta and MCTS implementations are to be submitted on **15 March, 6:00 p.m.**. The report and the contest agent must be submitted on **Friday 29 March, 6:00 p.m.**.
- *No delay* will be tolerated.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.
- The answer to questions must be given by filling in the latex template provided. The final file must be submitted on *gradescope*. No report sent by email will be accepted.
- Nothing must be modified in the template except your answer that you insert in the *answer* environments as well as your names and your group number. The names are provided through the command *students* while the command *group* is used for the group number. The dimensions of *answer* fields *must not* be modified either. For the tables, put your answer between the "&" symbols. Any other changes to the file will *invalidate* your submission.
- To submit on gradescope, go to `https://gradescope.com` and click on the "log in" button. Then choose the "school credentials" option and search for *UCLouvain Username*. Log in with your global username and password. Find the course LINFO1361 and the Assignment 2, then submit your report. Only one member should submit the report and add the second as group member.
- Check this link if you have any trouble with group submission `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members`

Deliverables

- The answers to all the questions have to be submitted in a report with the given format on Gradescope. Do not forget to put your group number on the front page.

- The following files are to be submitted on Inginious:
  - `alphabeta.py`: your implementation of the Alpha-Beta agent
  - `uct.py`: your implementation of the UCT MCTS agent
  - `agent.py`: your implementation of your contest agent

1 Exercises (5 points)

During the lectures, you have seen different adversarial search algorithms including the MiniMax and the Alpha-Beta algorithms. In this first section of the assignment, you will apply these two methods to a given example. You will also understand what heuristics and strategies can impact the performance of such algorithm.

Questions

1. Perform the MiniMax algorithm on the tree displayed in figure 1. In practice, you have to apply a value to each node (i.e. each red or green triangle). What move should the root player make? **(1 point)**
2. Perform the Alpha-Beta algorithm on the same tree. At each non-terminal node, put the successive values of α and β. Cross out the branches leading to non-visited nodes. We consider the tree from left-to-right here. **(1 point)**
3. Do the same exercise but assume a right-to-left lecture of the tree. **(1 point)**
4. Is there a node ordering that can lead to a more optimal pruning of the tree (in the sense where the algorithm prunes more branches than in the two other considered cases)? If no, explain why. If yes, give a new node ordering and the resulting new pruning. **(1 point)**
5. How can we modify Alpha-Beta algorithm for games with more than two players? **(1 point)**

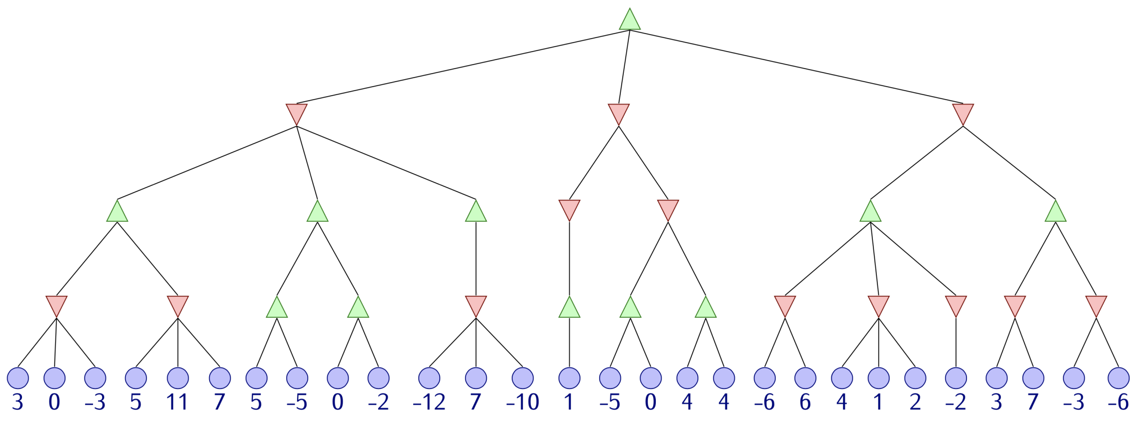

Figure 1: Tree for MiniMax and Alpha-Beta questions

2 Shobu (35 points)

We start now the main part of this project: the design of an AI agent able to play a game, Shobu. The rules of Shobu are available in the section 4. You may also watch this short Youtube video to have a quick explanation of the rules.

The code with the implementation of the game is available at `https://github.com/AchilleMorenville/LINFO1361-Shobu`. We strongly advise you to have a look at the different files to understand the data structures and how the game is implemented. You should not modify anything except for the different template files.

A random agent is given to you in the file *random_agent.py*

2.1 Alpha-Beta agent (4 points)

The first step in this section is to code a very basic Alpha-Beta[1] agent using the template *template_alphabeta.py*

Note that you should follow exactly the steps described here. Do not imagine any other evaluation or algorithm for this question. You will have the opportunity to implement whatever you want later in this project. For the moment, you don't need to worry about the *remaining_time* parameter in the *play* function.

Questions

1. Fill the **cutoff** function so that your agent stops at the prescribed maximum depth.

   The depth gives the level of the node in the alpha-beta tree where the cutoff criterion is being applied. Depth 0 means the root of the tree, depth 1 means the node resulting from applying one action and so on.

   Do not forget that if the game is over, the search also needs to be cut.

2. Implement the **eval** function. For now, we impose a simple heuristic that you have to implement: the score of the state is the difference between the minimal number of pieces of the player among all the boards minus the minimal number of pieces from the opponent among all the boards.

   Remember that the evaluation function should be relative to the player id and not to the current player.

3. Fill the **max_value** and **min_value** functions based on the pseudo-code provided in the AIMA book.

Once this is implemented, you have to submit your code on Inginious (`https://inginious.info.ucl.ac.be/course/LINGI2261/2024_Assignment_2_AlphaBeta`).

Each function will be tested independently to ensure the correctness of your alpha-beta agent. It will not yet play against another agent on Inginious (but you are free to test that on your own computer).

[1] Edwards, D. J., & Hart, T. P. (1961). The alpha-beta heuristic.

2.2 Monte-Carlo Tree Search agent (5 points)

Once your alpha-beta agent has passed all the tests on Inginious, consider the implementation of a Monte-Carlo Tree Search[2] (MCTS) agent using the template *template_uct.py*. More precisely, you will implement the UCT[3] version of the algorithm, which is MCTS which uses the UCB1[4] score as its tree policy.

Once again, stick to the steps and functions described below. You will have plenty of time later to look for funny things to implement. For the moment, you don't need to worry about the *remaining_time* parameter in the *play* function.

Questions

1. Implement the **UCB1** function that calculates the UCB1 score for a node in the search tree. This score balances exploration and exploitation, considering both the average reward obtained from visiting the node and the uncertainty associated with it. You can use the $\sqrt{2}$ for the C constant.

2. Fill the **select** function which start from the root node and select successive child nodes until a leaf node is reached. The root is the current game state and a leaf is any node that has a potential child from which no simulation has yet been initiated or when the game is finished. The selection chooses the child with the highest UCB1 score. For nodes that have not been visited yet, an infinite score should be assigned.

3. Implement the **select** function that starts from the root node and iteratively selects child nodes until reaching a leaf node. A **leaf** node is any node with a potential child from which **no simulation** have been run. During selection, the child with the **highest UCB1 score** is chosen. **Unvisited nodes** are assigned an **infinite score** to encourage exploration.

4. Implement the **expand** function that takes a node and adds a child node to the tree, returning the child. If the node is a **terminal state** (game ending), it returns itself.

5. Implement the **simulate** function that performs a **random simulation** starting from the given state and continues until the game ends. This function returns the **resulting utility** of the simulated game. Set a limit of 500 on the maximum number of rounds the simulation can run, as a random run can theoretically go on forever.

6. Implement the **back_propagate** function that takes the simulation result and updates the **visit count** and **win count** for each node on the path from the root to the previously selected child node.

   Important note: The win count stored in each node represents the **win rate for the player whose action led to that node**, not the player who will play from

[2]Browne, C. B., & et al., S. (2012). A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1), 1-43.

[3]Kocsis, L., & Szepesvári, C. (2006, September). Bandit based monte-carlo planning. In European conference on machine learning (pp. 282-293). Berlin, Heidelberg: Springer Berlin Heidelberg.

[4]Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. Machine learning, 47, 235-256.

that node in the future.

As before, you should submit your code on Inginious (`https://inginious.info.ucl.ac.be/course/LINGI2261/2024_Assignment_2_MCTS`). Due to the random nature of the MCTS algorithm, your agent will be evaluated on specific board positions and against random agents.

2.3 Your own agent (11 points)

Once you have implemented the two basic agents, alpha-beta and MCTS, you can start to think about your own agent that you will use for the contest.

2.3.1 Warm-up questions

Before directly diving into the code, consider the short questions below. They should help you think about different aspects of the algorithms that can be optimized for your agent.

Questions

1. What is the branching factor at the start of the game? What is the mean empirical branching factor? (The branching factor is considered here as the number of possible moves that a player can do from a given state). **(1 point)**
2. What would be one (of the many) drawbacks of the simple heuristic that is imposed for the basic alpha-beta agent above? **(1 point)**
3. Considering the Monte-Carlo Tree Search algorithm, what is the hypothesis supporting the use of random simulation to estimate the win rate? Is this hypothesis always valid? **(1 point)**

2.3.2 The real deal

From now on you have a free hand. You may continue to use alpha-beta or MCTS with improvements to the evaluation function or to the algorithms themselves. You can also use something entirely different and consider another algorithm that you may not have seen during the course. It's important to note that you'll now have to manage the *remaining_time* parameter, which indicates the remaining game time in seconds for your agent. If this remaining time drops to 0 before the end of the game, your agent will lose.

The important thing here is to describe what you have implemented and why you have chosen this particular approach. The grade for this part relies entirely on your explanation of your agent. Therefore you should explain all the tricks that you have coded and why you think they are good for the game of Shobu.

To guide you further for the report of this part, consider the following points that you could answer. You do **not** have to answer to a given point here if it does not apply to what you have coded for your agent! It is also very likely that you will have more to tell than what is listed here.

- Did you use another algorithm than alpha-beta or MCTS? If yes describe it.
- Did you improve in any way the alpha-beta or MCTS algorithms? How? Is it a specific approach for Shobu?

- Did you implement another evaluation function? How does it work? What is your intuition behind it? Explain it.
- Did you use any trick to speed up the search of your algorithm?
- ...

2.4 Comparison of agents (5 points)

In this last part, you are asked to perform a comparison of the agents that you use in this project.

This comparison should be a statistical one, **not** a theoretical one! You have to compare the following agents: *random*, *alpha-beta* (the basic version from above), *MCTS* (the basic version from above) and *your agent*. Feel free to do more comparisons with algorithms if it helps support the claims of the previous section about your agent

You are free to perform the experiments you want to complete this comparison but you are expected to detail the results in the report.

Additionally to these different experiments, you have to draw observations and conclusions based on the results you observe. Is one agent always better than the others? Why? Can you justify the choices you have made for your agent based on your comparison? And any other observation that you believe to be interesting.

You will be graded based on your method of performing the comparison but also on the quality of your observations and conclusions.

2.5 Contest (10 points)

Now, it is time for blood and glory.

Once you have implemented your own agent with everything that you wanted, submit it on Inginious (`https://inginious.info.ucl.ac.be/course/LINGI2261/2024_Assignment_2_Contest`). We will play a simple match with a random agent to ensure that your code runs without any problem.

Important: you have to submit only **1** file for your contest agent. This is non-negotiable. However, you may put everything needed inside this file. For your final submission, your names should be written as comments in the code to help identify your group.

For the evaluation, your agent will play against 3 of our agents with different levels: easy, medium and hard. You will receive 1 point for each vanquished agent. For each of these matches, we will run several games to ensure that you did not win only by chance.

The remaining 7 points will be decided by the contest with the other groups. Your agent will face the other ones from the other students. At the end, you will receive points based on your ranking obtained during the contest. The higher you are in the contest, the more points you will receive!

Each agent will be allowed 5 minutes playing time in all matches.

3 How to run the code

Different options are made available in the code to run the code.

You can simply launch your code by doing

```
python3 main.py
```

You can complete this simple line with different options:

- **-w {random|alphabeta|mcts|agent|human}**: white player agent
- **-b {random|alphabeta|mcts|agent|human}**: black player agent
- **-t <time value>**: time (in seconds) allowed to each player to complete the game
- **-d**: display option, shows a graphical interface for the game
- **-l <filename>**: log option, stores the game into the given filename
- **-r <filename>**: replay option, use the log file given as filename to replay the recorded game
- **-dt <value>**: delay time option, time (in seconds) between each move when replaying a game
- **-st <value>**: start turn option, the number of turn at which the replay should start

By using "-w human" and/or "-b human", you can play a game of Shobu through the graphical interface implemented in the code. You can simply select the pieces and the squares you want to move by clicking on them with your mouse. If you want to cancel a move, for example undo the passive move before doing the active move, you can simply press "u" (for "undo") on your keyboard. Note that Pygame may not respond directly to your input. If it is the case, simply hold the "u" key until it is accepted.

As an example, consider the following command:

```
python3 main.py -w human -b agent -t 600 -d -l logs.txt
```

In the order of the command: it will launch a Shobu game with you, a human (or any alien friend you may have, we do not want to discriminate them here) as the white player, your agent as the black player, each agent having 600 seconds to perform all their moves, displaying the game through the graphical interface and recording the game in a file named "logs.txt".

You can replay this game by using the following command:

```
python3 main.py -r logs.txt -dt 1 -st 42
```

In the order of the command: it will replay the game stored in the logs.txt file, pause during 1 second between each move and starts at turn 42 of the recorded game.

4 How to play Shobu

Shobu is a 2 persons strategic game composed of 4 boards, 2 lights and 2 darks, with 16 stones (also called pieces), 8 whites and 8 blacks, with 4 of each on every boards.

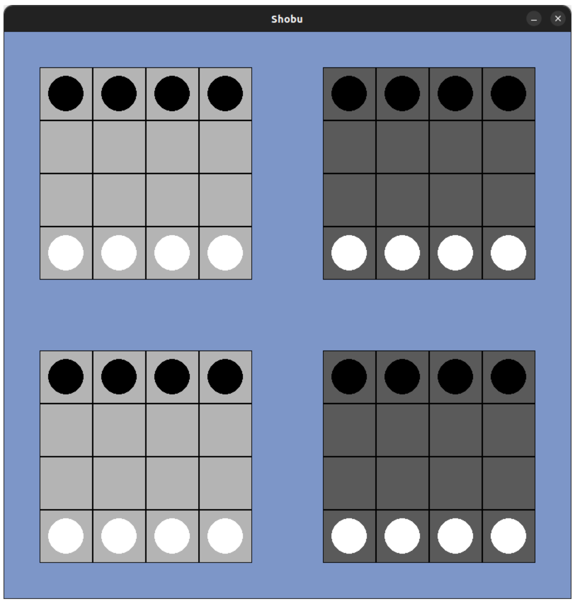

Figure 2: The setup at the start of a Shobu game

The boards are distinguished by their color but also with their relative position for each player. The two closest boards to you are called *home boards* while the two others are referred to as *opponent's boards*. This is illustrated from the white player's perspective in figure 3.

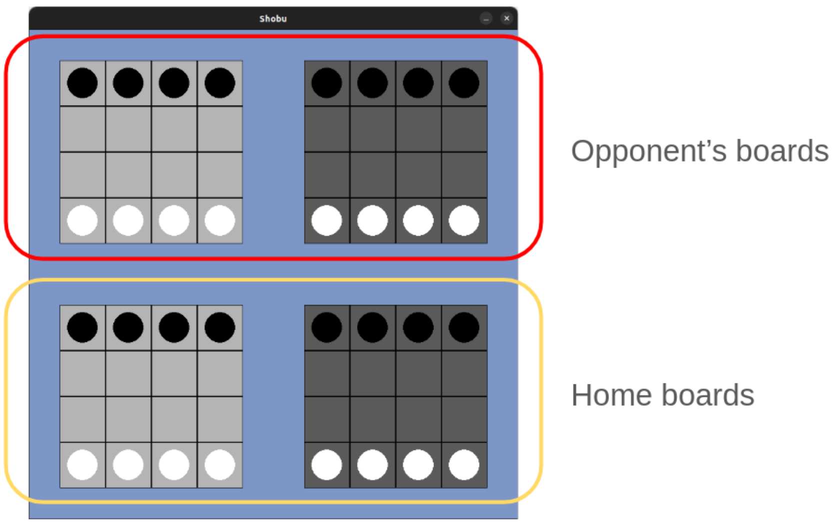

Figure 3: The home and opponent's boards when playing with the white pieces

How to move the pieces

One turn (also called action) is separated into 2 distinct moves: a passive move and an active move.

You begin your turn with the passive move. On one of your home boards, you can move one of your pieces in any direction (horizontally, vertically, diagonally) by 1 or 2 squares. This passive move is however constraint: you can not push or jump over any piece, yours or your opponent's. The piece you are moving can also not leave the board.

Next, you have to do an active move. This move is made on the board with the opposite color from the board where you played your passive move, i.e. light (resp. dark) if you did your passive move on your home dark (resp. light) board. You have to move one of your pieces with the exact same direction and length than during your passive move. However, the moving piece may be at a different starting position than the one used for the passive move (as long as it can indeed perform the move with the given direction and length). Note that your piece can still not leave the board.

It is during this active move that you may push **one** of the opponent's pieces. However, you can not push more than one of your opponent's pieces and you can not push any of your pieces. Thus, if two pieces are aligned with the direction of the active move, the move can not be performed. Any piece that is pushed out of the boards is considered out for the rest of the game.

Winning condition

The goal of this game is to push all of your opponent's pieces out of one of the four boards. It is also possible to win when the opponent can no longer make any valid move. An example where the black player wins is shown in figure 4.

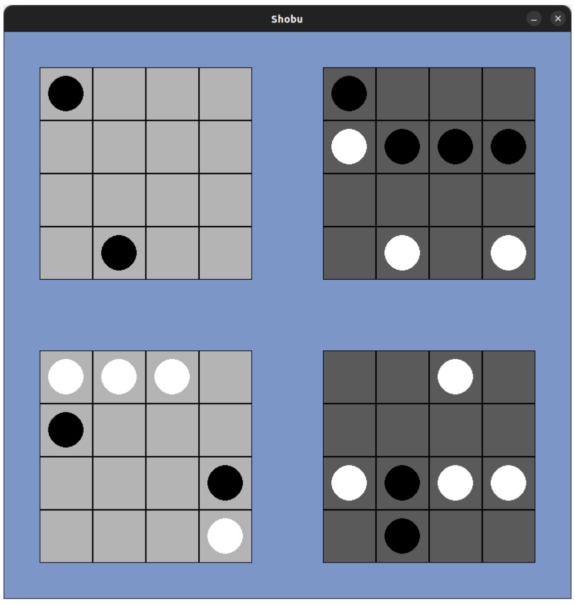

Figure 4: Example of a winning state for the black player. The top left board (also designated as the opponent's light board) does not contain any white piece, thus ending the game.

Figure 5: a very old representation of an early stage in the making of Shobu, a game invented in 2019