LINGI2261: Artificial Intelligence
Assignment 3: Adversarial Search

Alexander Gerniers, Gael Aglin, Yves Deville
March 2021

Guidelines

- This assignment is due on **Thursday 1 April 2021 at 6pm for the first part and on Thursday 22 April 2021 at 6pm for the second part**.
- This project accounts for **40% of the final grade for the practicals**.
- *No delay* will be tolerated.
- Not making a *running implementation* in *Python 3* able to solve (some instances of) the problem is equivalent to fail. Writing some lines of code is easy but writing a correct program is much more difficult.
- *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.
- Indicate clearly in your report if you have *bugs* or problems in your program. The online submission system will discover them anyway.
- 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. A specific submission procedure for the *contest* will be detailed later.
- 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. 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 UCLouvain email and password. Find the course LINGI2261 and the Assignment 3, then submit your report. Only one member should submit the report and add the second as group member.
- For those who have not been automatically added to the course, at the right bottom of gradescope homepage, ckick on "Enroll on course" button and type

the code *86KJVB*.

- 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 in a report **on Gradescope.**
- The following files are to be submitted on *INGInious* inside the *Assignment 3* task(s):
  - **All the code that you need for this assignment provided in a Git.** Make sure to keep it updated as there might be some updates:

    `https://github.com/MIFY-AI/MAIC2020-Seega`

    If you don't know how to use git you can simply download the folder with the web interface from time to time. The repository contains a `README.md` file that explains how to launch the GUI of the game. Once the board is displayed, you can launch the match by clicking on *Game* > *New Game* (or Ctrl-N / Cmd-N).
  - `basic_agent.py`: the basic Alpha-Beta agent from section 2.1. It should be submitted on task *Assignemnt 3 - Basic Agent*. The file must be encoded in **utf-8**.
  - `smart_agent.py`: your smart alpha-beta agent from section 2.5. It should be submitted on task *Assignemnt 3 - Smart Agent*. The file must be encoded in **utf-8**.
  - `contest_agent.py`: your super tough agent that will compete in the contest. Instructions can be found in section 2.6. Specific instructions regarding the submission of your contest agent will follow later on. The file must be encoded in **utf-8**.
- Please note that your `basic_agent.py` submission to the associated INGInious task is due for **Thursday 1 April 2021 at 6pm** . After that date, the associated INGInious task will be closed and you won't be able to submit your basic alpha-beta agent.
- A **mid-project Q&A session** will be organized shortly after the first deadline. You should *at the very least* have done sections 1 and 2.1 (as said above, section 2.1 must be completed on INGInious for the first deadline). For this session, come with questions about your smart agent or about the contest.
- The `smart_agent.py` and `contest_agent.py` files are due on **Thursday 22 April 2021 at 6pm** on INGInious.

1 Alpha-Beta search (5 pts)

During lectures, you have seen two main adversarial search algorithms, namely the MiniMax and Alpha-Beta algorithms. In this section, you will apply and compare by hand these two algorithms. You will also understand what heuristics and strategies can impact the performances of these algorithms.

The small example provided in Figure 1 differs a bit from the ones provided in the lecture slides. As is the case for the Seega game that you will play with in the rest of the assignment, there is no strict alternance between players: some situations lead to a state where the same player may make the next move. In the diagram, "max" nodes will be indicated with green triangles going up, and "min" nodes with red triangles going down.

Questions

1. Perform the MiniMax algorithm on the tree in Figure 1, i.e. put a value to each node. Circle the move the root player should do.
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 arcs reaching non visited nodes. Assume a left-to-right node expansion.
3. Do the same, assuming a right-to-left node expansion instead.
4. Can the nodes be ordered in such a way that Alpha-Beta pruning can cut off more branches (in a left-to-right node expansion)? If no, explain why; if yes, give the new ordering and the resulting new pruning.
5. How does Alpha-Beta need to be modified for games with more than two players?

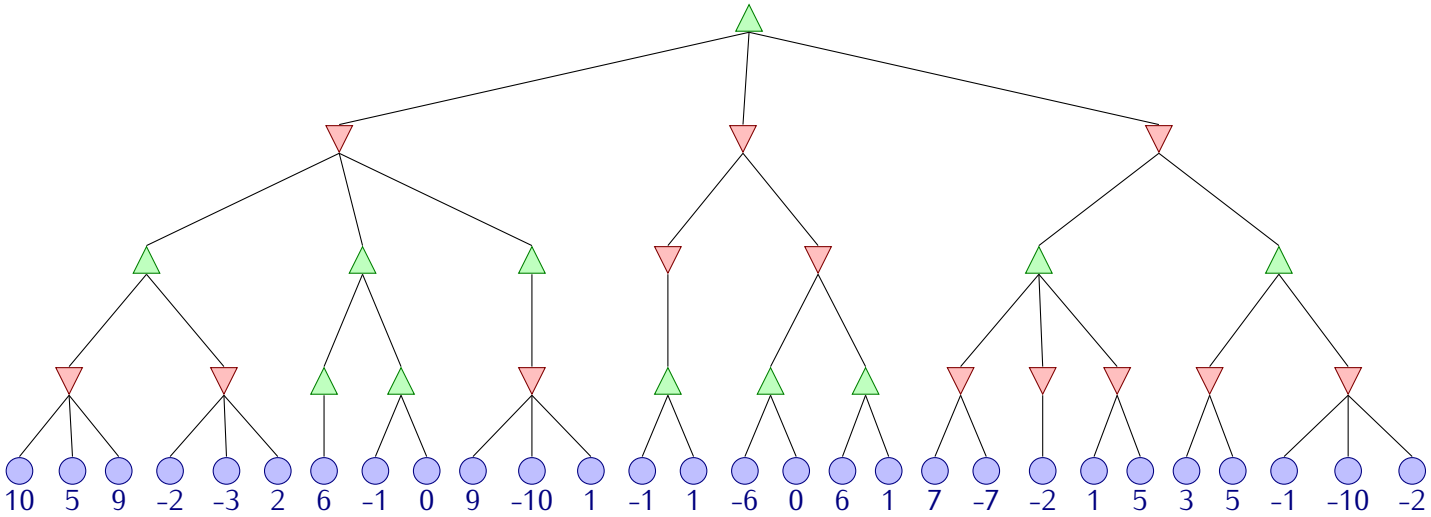

Figure 1: MiniMax

2 Seega (35 pts)

You have to imagine and implement an AI player able to play the Seega game. The rules of this game can be found on `http://www.cyningstan.com/game/120/seega`. The goal is to capture all the opponent's pieces. Note that we will follow the rule indicated at the bottom of the web page, stating that in case of a draw (i.e. nobody is able to capture an opponent piece anymore), the player with the most remaining pieces wins. There are thus 2 different ways to win the game! Before you start the rest of this assignment, you should read carefully these rules.

Attention

Throughout this assignment you will have to implement your agent by extending the `Player` class provided in `core/player.py`. Your class **must** be called `AI`.

The template of an Alpha-Beta agent is provided in `template_agent.py`, where you only need to fill in the gaps in order to do an Alpha-Beta agent. This template file is provided together with these instructions (not on the Git repository).

The state of the Seega game is represented by the `State` object implemented in `seega/seega_state.py`, itself using the `Board` object of `core/board.py` to represent the current state of the board. The status of a particular cell is given by the `Color` object (defined in `core/player.py`), indicating which player (green or black), if any, has a piece at that position.

The rules of the game, including functions that verify if actions are legal, are implemented in `seega/seega_rules.py`.

2.1 A Basic Alpha-Beta Agent (5 pts on INGInious)

Before you begin coding the most intelligent player ever, let us start with a very basic player using pure Alpha-Beta. Clone the git repository and fill in the gaps of the file `template_agent.py` by answering to the following questions.

Note: you are required to follow **exactly** these steps. Do not imagine another evaluation function for example. This is for later sections. You don't need to write anything in the report about this part.

Questions

1. Fill in the `successors` function. You can use the function:

   ```
   SeegaRues.get_player_actions(state, player)
   ```

   to get all the actions of a certain player. You need to yield each action together with the resulting state after the action is applied. This state is obtained by:

   ```
   SeegaRules.act(state, action, player)
   ```

   Do not forget that you need to copy the state before you apply the action otherwise you will apply all actions to the input state. You can do this with the

function `copy.deepcopy(state)`.

2. Define the `cutoff` function so that your agent greedily selects the best actions. By this we mean that of all the possible actions it can do, it should select the one that maximizes the evaluation function.

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

   Don't forget that if the game is over, the search also needs to be cut. You can check this using the function `SeegaRules.is_end_game(state)`.

3. Implement the `evaluation` function to be the number of opponent pieces the player has managed to capture.

   Remember that the evaluation function should always be relative to your player color/id, not the current player. The player color is defined by the `Color` object in `self.color`, and the corresponding id (1 for the green player, −1 for the black player) can be accessed by `self.color.value` (or `self.position`). The other player id is obviously given by `-self.color.value`.

4. Upload your solution in the INGInious task *Assignment 3 - Basic Agent*. Note that this submission is due before **Thursday 1 April 2021 at 6pm** .

Now you will begin to implement a smarter Alpha-Beta agent. In the next sections, the questions are there to help you to create a very smart agent. We highly recommend you answer these questions before coding your smarter alpha-beta agent since they will help you in this process. Note that for now, you are still restricted to use an Alpha-Beta agent based on the `template_agent` file.

2.2 Evaluation function (5 pts)

When the size of the search tree is huge, a good evaluation function is a key element for a successful agent. This function should at least influence your agent to win the game. A very simple example was implemented in the basic agent. But we ask you to make a better one.

Questions

5. What are the weak points of the evaluation functions of your basic agent?
6. As described in the class, an evaluation function is often a linear combination of some features. Describe new features of a state that can be interesting when evaluating a state of this game.
7. Describe precisely your improved evaluation function.

Attention

You should describe in this part the best evaluation function you can come up with. This question is not related to whether your smart agent passes the tests on *Inginious* (see question 16). This means that if your agent passes the *Inginious* tests but you further improve your evaluation function later on, you should describe here this latest version.

2.3 Successors function (4 pts)

The successors of a given board can be obtained by applying all the moves returned by the `get_player_actions` method. However, there might be too many of them, making your agent awfully slow. Another crucial point is thus designing a good way to filter out some successors.

Questions

8. Give an upper bound on the maximum number of actions that can be performed in any given state. If you can't, explain how you could estimate what is the average number of successors and estimate it.
9. What do you loose by ignoring some of the successors?
10. Could reordering the successors help the performance of your agent? If so, in what order should the successors be returned in order to help Alpha-Beta prune the search tree?
11. Describe you successor function.

2.4 Cut-off function (4 pts)

Exploring the whole tree of possibilities returns the optimal solution, but takes a lot of time (the sun will probably die before, and so shall you). We need to stop at some point of the tree and use an evaluation function to evaluate that state. The decisions to cut the tree is taken by the cut-off function. **Remember that the contest limits the amount of time your agent can explore the search tree.**

Questions

12. The `cutoff` method receives an argument called `depth`. Explain precisely what is called the *depth* in the `minimax.py` implementation.
13. The `cutoff` function is also the one responsible for identifying which states lead to the end of the game (and so possible victory or defeat). One such situation is a draw, called a `boring` state in the implementation, where each player has managed to build a "barrier" behind which pawns can move indefinitely without risking to be attacked. In the implementation, this is dealt with by ending the game when 200 boring moves have occurred. Why could such an approach be

problematic for an Alpha-Beta agent? How could you remedy this?

14. In the Seega contest your agent will be credited a limited time. How do you manage that time? How can you make sure that you will never timeout? Explain how you can use *iterative deepening* to avoid timeouts.
15. Describe your cut-off function.

2.5 A Smart Alpha-Beta Agent (6 pts on INGInious)

Using the answers from Section 2.4 and Section 2.2, build a smart alpha-beta agent. Now the time has come to make it play against a simple alpha-beta agent to see how much your agent improved.

Attention

Such an agent must still follow the `template_agent` and only implement changes that are discussed in the preceding questions (i.e. changes to the evaluation, successor and cutoff functions, as well as possible changes to implement iterative deepening). Save all your crazy ideas for the contest ;-).

Questions

16. Upload your agent to the **3** INGInious tasks named *Assignment 3 - Smart Agent (match x)*. Your agent will play 3 matches against different players (including a basic alpha-beta agent). For each match, each agent will have a time credit of 5 minutes. Your score to this question will be proportional to the number of agents you're able to beat.

2.6 Contest (6 pts on INGInious; 5 pts for report)

Now that you have put all the blocks together, it is time to set you free! You may improve your agent in order to build a super tough competition agent. You are free to tune and optimize your agent as much as you want, or even rewrite it entirely. You are not restricted to Alpha-Beta if you wish to try out something crazy.

Your agent will fight the agents of the other groups in a pool-like competition. We do emphasize on two important things:

1. *Technical requirements* must be carefully respected. The competition will be launched at night in a completely automatic fashion. If your agent does not behave as required, it will be eliminated.
2. Your agent should not only be smart, but also *fair-play*. Launching processes to reduce the CPU time available to other agents, as well as using CPU on other machines are prohibited. Your agent should run on a single core machine. Any agent that does not behave fairly will be eliminated. If you are in doubt with how fair is a geeky idea, ask us by mail. Your agent must only be working during the calls to `play`.

Questions

17. Specific instructions for the upload of your contest agent will be given later on.
18. Describe concisely your super tough agent in your report. **Your description shouldn't be longer than two A4 paper pages.**.

Attention

Any improvements you possibly made regarding the evaluation, successor and cutoff functions that could also apply to a smart agent should be explained in the corresponding questions (sections 2.2, 2.3 and 2.4). Here, you should detail the improvements you made that go *beyond* the scope of these sections. The 5 points assigned to question 18 will thus be awarded for testing and implementing *other* ideas.

Some good ideas on paper may not work out in practice... and that's okay! If you tested something that seemed promising but didn't give the expected results, had a low benefit/cost ratio, etc., mention it in your report. Explain why you thought it was promising, what where your test results, and why you decided not to use it.

Technical requirements

- Your agent must extend the class `Player` provided in `core/player.py` and the class must be called `AI`.
- You need to put the following name in the `name` class variable of your agent: `Group XX` where `XX` is your group number.
- The only other function you need to implement is `play`. You can do whatever you want in this function but the result must be an action, i.e. an instance of the `Action` class.
- You are allowed to implement your own representation of a state if you deem it useful.
- Only agents being able to defeat the dummy alpha-beta player will be allowed to compete.
- Your agent should be implemented in a file named `contest_agent.py`.
- Your agent must use the CPU only during the calls to the `play` method. It is forbidden to compute anything when it should be the other agent's turn to play.
- You can assume your agent will be run on a single-processor single-core machine. Do not waste time working on parallelisation of your algorithms.
- Your agent cannot use a too large amount of memory. The maximal limit is set to 4 Gb. We do not want to look for a super computer with Tb's of memory just to make your agent running.
- Your agent will receive a time credit for the whole game. The time taken in the `play` method will be subtracted from this credit. If the credit falls below 0, your agent will be disabled and random moves will be performed instead. In any case, you should avoid timing out at all cost! The time credit is passed by as argument of the `play` method.

Contest rules

The contest will be played with the Seega rules described in the Seega Instructions file provided on Moodle. Each agent will get a time credit of **15 minutes** per match.

For the first phase of the tournament, the agents will be divided into different pools. All agents inside one pool will play twice (once playing as first player and once playing as second player) against all other agents in the same pool. For each match, the winner gets 3 points. In each pool, the two agents having the highest number of points will be selected. In case of ties, the slowest agent will be eliminated. The x best third-place agents will also be selected to obtain the 16 agents of the final phase.

The selected 16 agents will participate in a knock-out phase, leading to a final. For each duel, 2 game will be played (such that each agent plays once as first player). For each duel, the winner is the agent winning more games than his opponent. If both agents win 1 match, then the fastest player will be selected.

The trace of each match will be stored and the most exciting or representative ones will be replayed during the debriefing sessions.

This year, you will not only compete among yourselves, but you will also battle with agents submitted from around the world! Will your agent be crowned world champion of the Seega game ?!

Evaluation

The mark you will get for this part will be based on:

- the quality of your agent,
- the quality of your documentation,
- the originality of your approach,
- the design methodology (i.e., how did you chose your parameters?).

The position of your agent in the contest will give you bonus points.

Licensing

The provided classes are licensed under the General Public License[1] version 3. Your agents shall also have a GPL license.

May the Force be with you!

We hope your agents will play exciting games and that they will outsmart the humans. We hope you will have these small amounts of luck needed to make good thoughts turn into great ideas.

[1] `http://www.gnu.org/copyleft/gpl.html`