LINGI2261: Artificial Intelligence
Assignment 3: Adversarial Search

Alexander Gerniers, Gael Aglin, Yves Deville

November 2019

Guidelines

- This assignment is due on **Wednesday 13 November 2019 at 6pm for the first part and on Wednesday 27 November 2019 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. The consequences of *plagiarism* is *0/20 for all assignments*.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No report or 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*.

Deliverables

- The answers to all the questions in a report **on INGInious. Do not forget to put your group number on the front page as well as the INGInious id of both group members.**
- 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://bitbucket.org/agerniers/squadro_public/`

    If you don't know how to use git you can simply download the folder with the web interface from time to time. You should read the task *Assignment 3 - Getting started* before you start. It contains the documentation of the code as well as the git commands to copy the code to your machine.

  – `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**.

  – `super_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. You will submit your agent on task *Assignemnt 3 - Contest Agent*. The file must be encoded in **utf-8**.

- Please note that your `basic_agent.py` submission to the associated INGInious task is due for **Wednesday 13 November 2019 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 session** will be organized shortly after the first deadline. Details on this session will be provided later. The attendance at this session is mandatory. During this session, we will discuss strategies to get a strong alpha-beta agent. You should at least have done sections 1 and 2.1 (as said above, section 2.1 must be completed on INGInious for the first deadline (13 November)). For this session, come with questions about your super agent or about the contest.

- The `smart_agent.py` and `contest_agent.py` files are due on **Wednesday 27 November 2019 at 6pm** on INGInious.

1 Alpha-Beta search (6 pt)

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.

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 tree in Figure 2. 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 (Figure 3).
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?

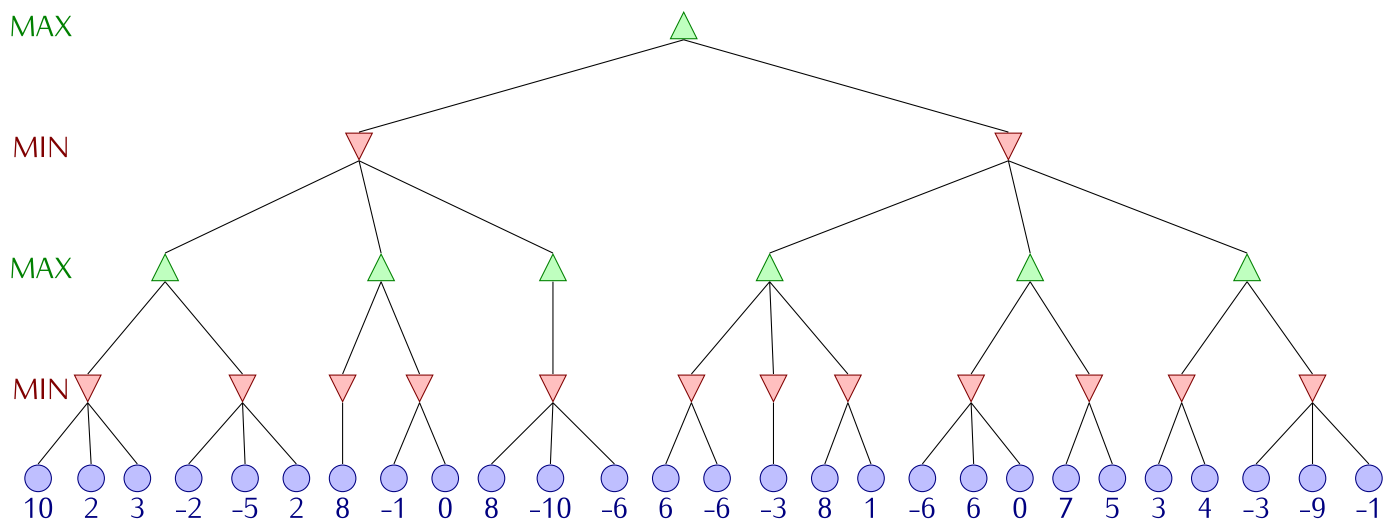


Figure 1: MiniMax

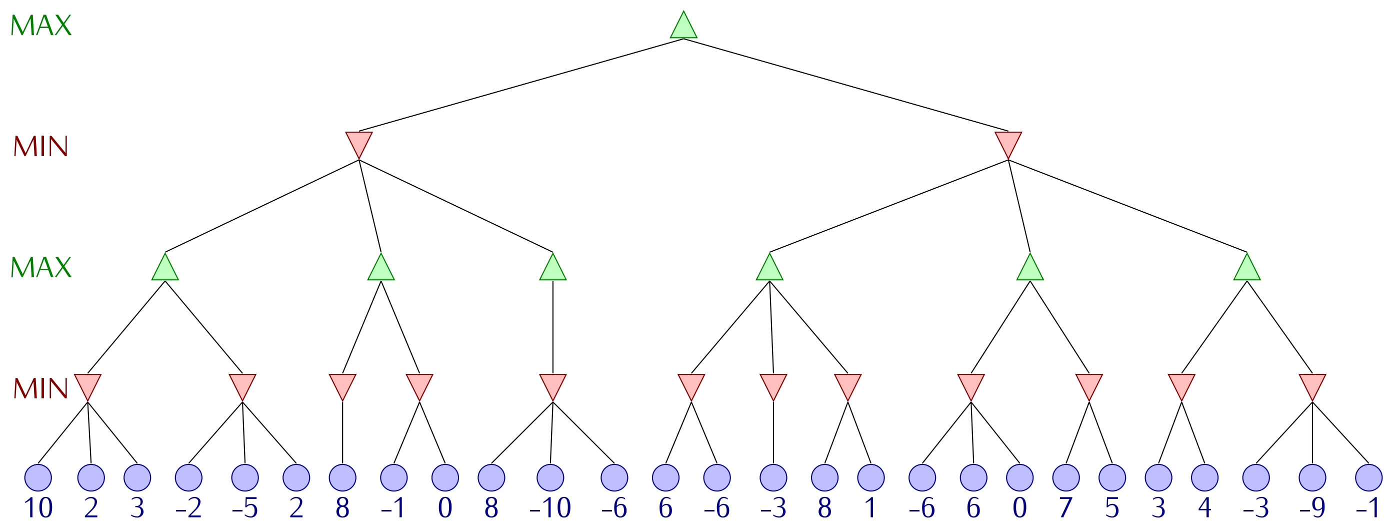


Figure 2: Alpha-Beta, left-to-right expansion

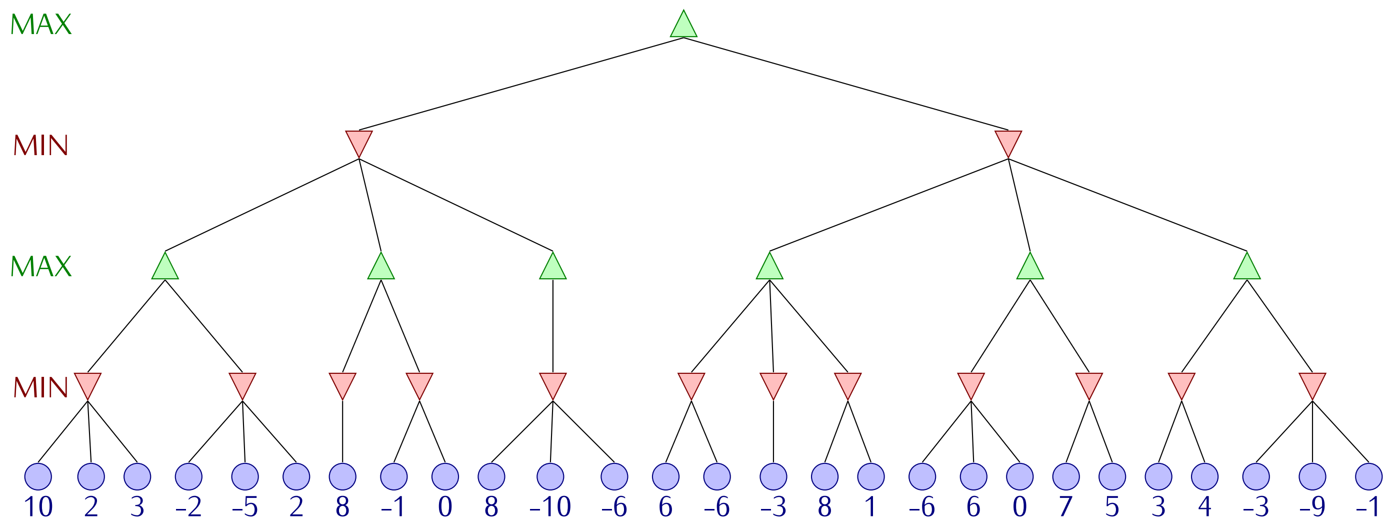


Figure 3: Alpha-Beta, right-to-left expansion

2 Squadro (34 pts)

You have to imagine and implement an AI player able to play the Squadro game. A copy of the rules is available on Moodle. Before you start the rest of this assignment, you should read the rules and the task *Assignment 3 - Getting started* from INGInious.

2.1 A Basic Alpha-Beta Agent (5 pts)

Attention

Throughout this assignment you will have to implement your agent by extending the Agent class provided in `agent.py`. Your class **must** be called `MyAgent`.

A template agent is provided in `template_agent.py` where you only need to fill in the gaps in order to do an Alpha-Beta agent.

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 get the actions from the current player by using the function `state.get_current_player_actions()` to get all the actions of the current 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 `state.copy()`.

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 `state.game_over_check()`.

3. Implement the `evaluation` function to be the sum of the advancement of each of the agent's pawns (for now, we are not taking into account the opponent). By advancement, we mean the number of tiles a pawn has advanced since the start, which can be obtained using `state.get_pawn_advancement(player_id, pawn)`.

   Remember that the evaluation function should always be relative to your player id, not the current player.

The player id represents whether you are the first or second player. You can access you player id (0 or 1) with `self.id`. The other player id is obviously given by `1 - self.id`.

4. Upload your solution in the INGInious task *Assignment 3 - Basic Agent*. Note that this submission is due before **Wednesday 13 November 2019 at 6pm** .

2.2 Comparison of two evaluation functions (3 pts)

Let us observe how the basic agent behaves when playing against itself while we slightly modify its evaluation function.

Questions

5. Make a second version of your basic agent which is an exact copy of the first but where the evaluation function takes into account the opponent: it should return the difference between the player's total advancement and the total advancement of its opponent.
6. Which evaluation function do you expect to perform better? Make your 2 basic agents compete each other in the 4 possible starting configurations (which color, who plays first). Do the results match your expectations? Explain.

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.

2.3 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

7. The `cutoff` method receives an argument called `depth`. Explain precisely what is called the *depth* in the `minimax.py` implementation.
8. Does your answer to question 6 change when increasing the cutoff depth (try with depths 3, 5, 8 and 10)? Explain.
9. In the Squadro 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.
10. Describe your cut-off function.

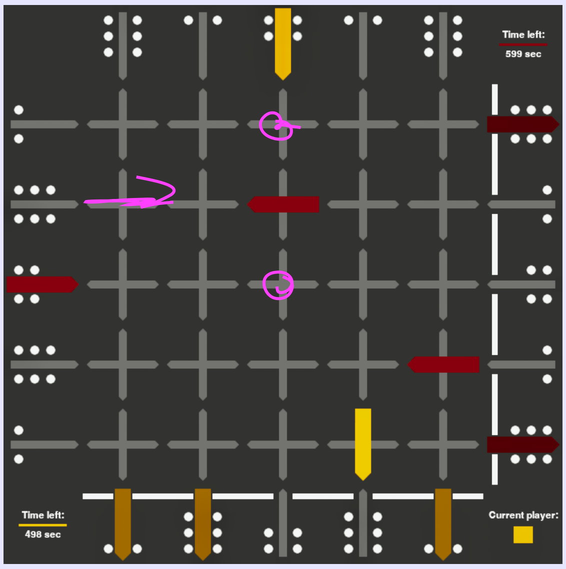
2.4 Evaluation function (4 pts)

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

11. Using the sum of each pawn's advancements as evaluation function can lead to strange behaviour. Have you spotted some? Explain why this happens and how you would change the evaluation function to prevent this from happening.

    Hint: what would your first basic agent (in yellow) play next in this situation?



12. Are there any other weak points of the evaluation functions of your basic agents?
13. 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.
14. Describe precisely your improved evaluation function.

2.5 A Smart Alpha-Beta Agent (8 pts)

Using the answers from Section 2.3 and Section 2.4, 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.

Questions

15. Upload your agent to the INGInious *Assignment 3 - Smart Agent* task. Your agent will play a match against a simple alpha-beta player. Each agent will have a time credit of 5 minutes. If you win against this agent, you get all the points for the question. If you get beaten, you don't get any point.

2.6 Contest (10 pts)

Now that you have put all the blocks together, it is time to assess the intelligence of your player. Your agent will fight the agents of the other groups in a pool-like competition. You are free to tune and optimize your agent as much as you want, or even rewrite it entirely. For this part, you are not restricted to Alpha-Beta if you wish to try out something crazy. 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

16. Upload your agent to the INGInious *Assignment 3 - Contest Agent* task. Your agent will play a match with a time credit of 5 minutes against a basic alpha-beta player. This is just to check that your agent is able to play. For the real contest, your agent will have a time credit of 15 minutes.
17. Describe concisely your super tough agent in your report. **Your description shouldn't be longer than two A4 paper pages.**.

Technical requirements

- Your agent must extend the class `Agent` provided in `agent.py` and the class must be called `MyAgent`. Note that you do not have to extend the `AlphaBetaAgent` if you do not want.
- You need to implement the function `get_name` on your agent an return the following name: `Group XX` where `XX` is your group number. For instance, group 1 will return `Group 01` and group 11 will return `Group 11`.
- The only other function you need to implement is `get_action`. You can do whatever you want in this function but the result must be an action. Actions are simply an integer (between 0 and 4) that indicates which pawn will move.
- Only agents being able to defeat the dummy alpha-beta player in the INGInious *Assignment 3 - Contest Agent* task 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 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 lose the game. The time credit is passed by as argument of the `get_action` method.

Contest rules

The contest will be played with the Squadro rules described in the Squadro Instructions file provided on Moodle.

Each agent will get a time credit of 15 minutes per match.

The agents will be divided into 7 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 two best third-place agents will also be selected to obtain the 16 agents of the final phase.

The selected 16 agents will participate to a best-of-four playoff. 4 games form a match (twice playing as first player and twice playing as second player). For each match, the winner is the agent winning more games than his opponent. If both agents win 2 matches then the fastest player will be selected. The selected agents will compete in matches as shown in Figure 4. The player labels displayed represent the pool from which the agent comes (i.e. pool A, B, C, D, E, F, G) and its position in this pool. The two best third place agents are denoted T1 and T2. A third place playoff will also be played to determine the third place between the losers of the semi-final.

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

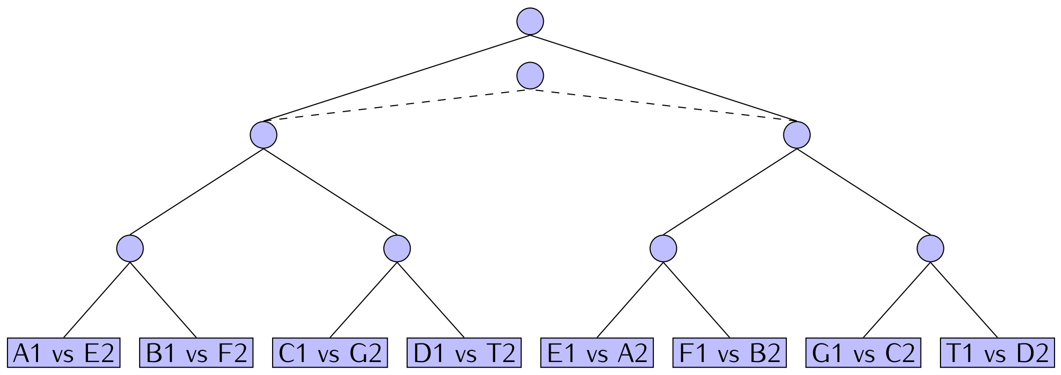


Figure 4: Playoffs

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. The working agents will then be put together and published on the website `http://becool.info.ucl.ac.be/aigames/squadro2019`.

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`