LINGI2261: Artificial Intelligence
Assignment 3: Project: Adversarial Search

François Aubry, Cyrille Dejemeppe, Yves Deville
October 2015

Guidelines

- This assignment is due on **Wednesday 4 November 2015 at 6 p.m. for the first part and on Wednesday 18 November 2015 at 6 p.m. 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*.
- Answers to all the questions must be delivered at the INGI *secretary* (paper version). Put your names *and your group number* on it. Remember, the more concise the answers, the better.
- 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 paper format **at the secretary**. **Do not forget to put your group number on the front page**.
- The following files are to be submitted on *INGInious* inside the *Assignment 3* task(s):
  - `basic_agent.py`: the basic Alpha-Beta agent from section 2.1. Your program should respect the skeleton provided on the Moodle website in the python 3 Avalam framework. The file must be encoded in **utf-8**.
  - `super_agent.py`: your smart alpha-beta Avalam AI agent. Your program should respect the API provided in the python 3 Avalam framework. The file must be encoded in **utf-8**.
  - `contest_agent.py`: your super tough Avalam AI agent that will compete in the *INGI2261 Avalam Tournament*. Your program should respect the API provided in the python 3 avalam framework. The file must be encoded in **utf-8**.
- A **mid-project session** will be organized two weeks before the deadline. During this session, we will discuss strategies to get a strong alpha-beta agent. The attendance at this session is mandatory (read: we will remove points if you do not come). You should at least have done sections 1, 2.1 and 2.2. Come with the questions you have about your super agent or about the contest.
- Please note that your `basic_agent.py` submission to the associated INGInious task is due for **Wednesday 4 November 2014 at 6 p.m.**. After that date, the associated INGInious task will be closed and you won't be able to submit your basic alpha-beta agent.

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 hands 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.

4: Yes, best solutions first

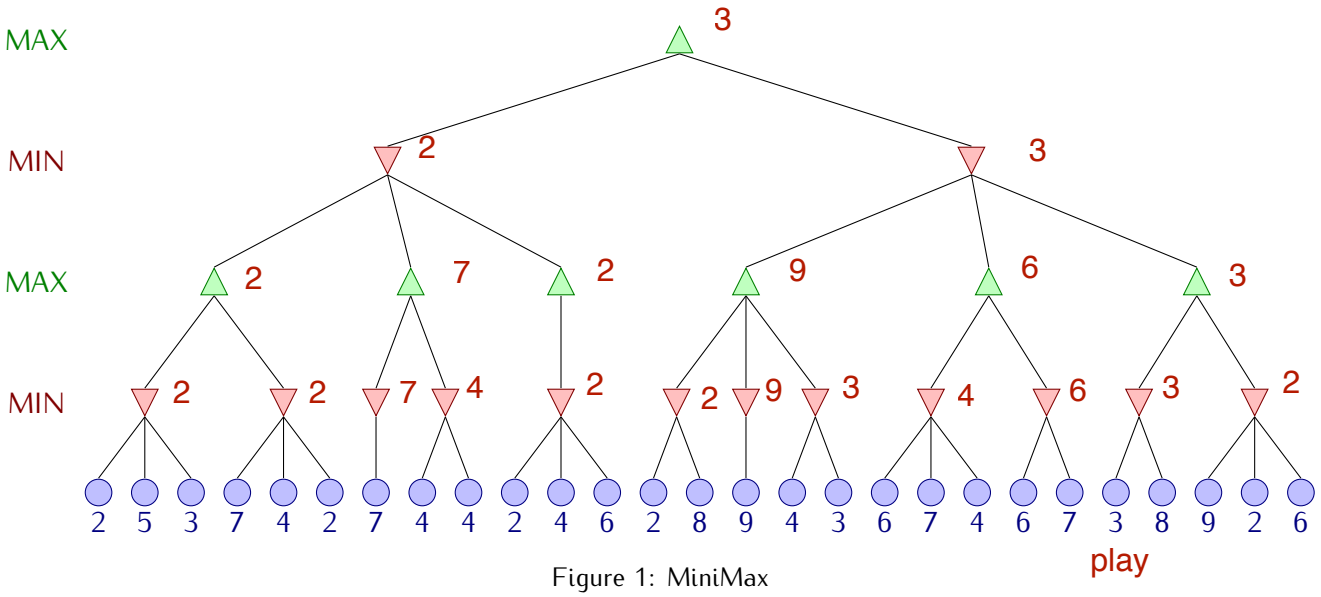

Figure 1: MiniMax

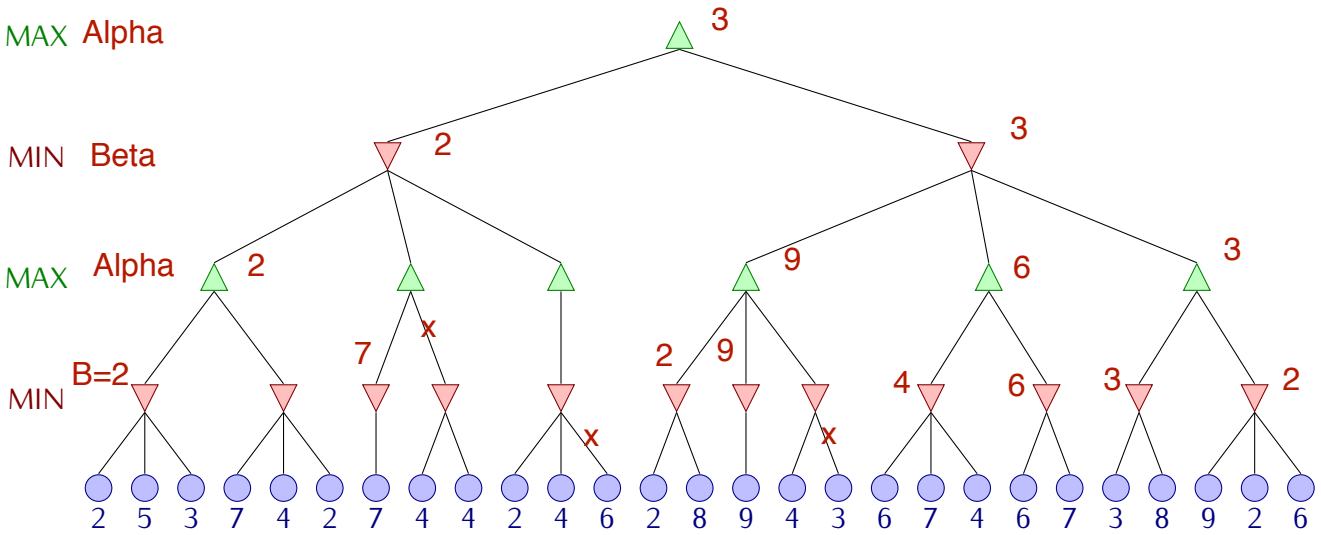

Figure 2: Alpha-Beta, left-to-right expansion

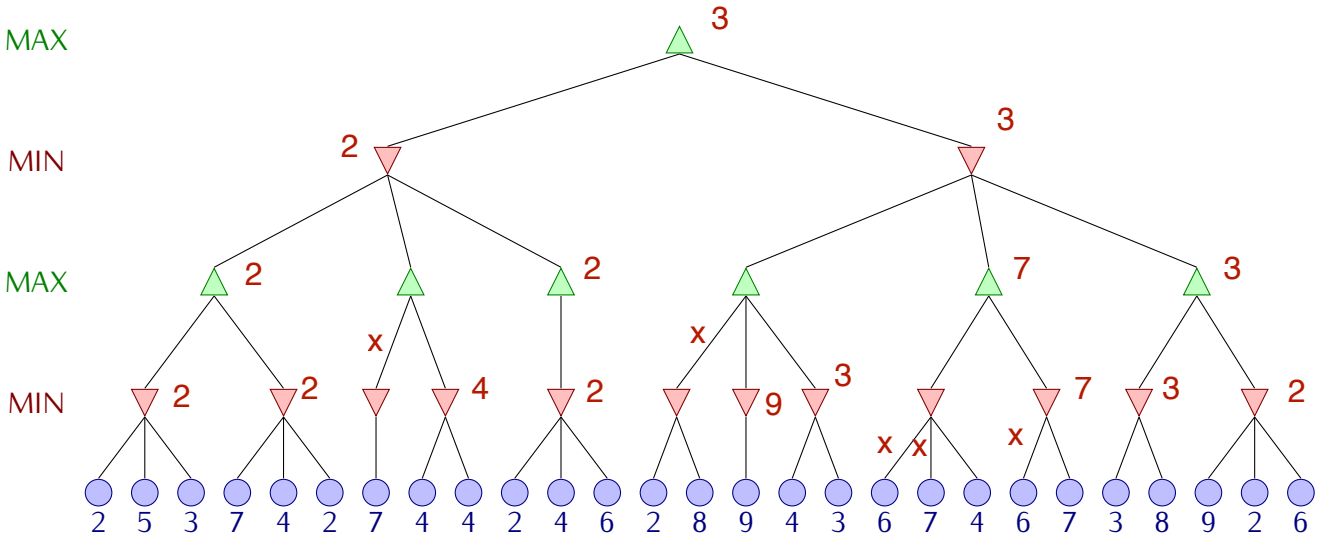

Figure 3: Alpha-Beta, right-to-left expansion

2 Avalam (34 pts)

You have to imagine and implement an AI player to play the Avalam game. A copy of the rules is available on the Moodle course website.

2.1 A Basic Alpha-Beta Agent (5 pts)

Before you begin coding the most intelligent player ever, let us start with a very basic player using pure Alpha-Beta. Copy the file `agent.py` to `basic_agent.py` and fill in the gaps 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.

Questions

1. Implement the `successors` method *without modifying* `minimax.py`. You shall return all the successors given by `Board.get_actions` in that order (i.e. all possible placements and all possible moves).
2. In order to avoid to explore the whole tree, we will limit the search up to a depth of 2. Of course, we have to ensure the search will also stop if the game is over. This can be done using the `Board.is_finished` method. Implement these ideas in the `cutoff` method.
3. The evaluation function you will use is a very simple one. This function shall return 1 if the agent is in an advantageous situation, -1 if his opponent has the advantage and 0 if no player has an advantage on the other. To determine whether an agent has the advantage, use `Board.get_score`. It returns the difference between the number of tower of the player and the number of towers of its opponent. If this difference is null, it returns the difference between the number of towers of height of the player and those of height 5 of its opponent Be careful that the evaluation function must always be evaluated for the player that has to play a move at the root of the search tree (i.e. even if you are at an opponent node on the search tree).
4. Upload your solution in the INGInious task *Assignment 3 - Avalam: Basic Agent*. Note that this submission is due before **Wednesday 4 November 2015 at 6 p.m.**.

2.2 Comparison of two evaluation functions (2 pts)

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

Questions

5. Starting from your basic agent implemented before, construct a new agent which is its copy except that its evaluate method returns directly the result of `Board.get_score` instead of -1, 0 or 1.
6. Launch a game where one of the agents is the basic agent described before against another agent where the basic evaluate method has been replaced such that it returns directly the result of `Board.get_score` instead of -1, 0 or 1. Watch the replay of the match. What do you observe? Does one of the agent clearly overcomes the other one? Explain why there is such a difference.

6: fail of the first basic agent, because he has
no informations if he start loosing or winning.

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 Evaluation function (2 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 is given by the `get_score` method of the `Board` class. It returns the difference between the number of tower with your colour with the number of towers with your opponent colour on the top. If the difference is null, it returns the difference between the number of towers of hight 5 of your colour and the number of towers of height 5 of your opponent colour. But we ask you to make a better one.

Questions

7. Describe precisely your evaluation function.

2.4 Successors function (4 pts)

The successors of a given board can be obtained by applying all the moves returned by the `get_actions(player)` 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 and a lower bound on the branching factor for a search tree on the Avalam game. Justify your answer.
9. How does the branching factor evolve after a move has been performed? Explain.
10. Can you think of states that might be ignored? What do you loose if you ignore successors? Explain.
11. Describe your successors function.

2.5 Cut-off function (3 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. Explain why it might be useful (for the Avalam contest) to cut off the search for another reason than the depth. Would it be interesting to consider a changing cutoff function (i.e. it changes according to the advancement of the game).
14. Describe your cut-off function.

2.6 A Smart Alpha-Beta Agent (8 pts)

Using the answers from Section 2.4 to Section 2.5, you built 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 - Avalam: Super Agent* task. Your agent will play a match against a simple alpha-beta player. Each agent will have a time credit of `2.5` minutes. If you win against this agent, you get all the points for the question. If you succeed to perform a draw, you get half of the points and if get beaten, you don't get any point. Don't forget to include all the needed files inside the archive you upload.

2.7 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 - Avalam: Contest Agent* task. Your agent will play a match against a basic alpha-beta player. Each agent will have a time credit of 2.5 minutes (for the real contest, your agent will have a time credit of 20 minutes per match). If you win against this agent, you get all the points for the question. If you succeed to perform a draw, you get half of the points and if get beaten, you don't get any point. Don't forget to include all the needed files inside the archive you upload.
17. Describe concisely your super tough agent in your report. **Your description shouldn't be longer than one A4 paper page; if it is the same agent than the one described in earlier sections, just state it, no need to re-explain.**.

Technical requirements

- The files needed by your agents should be submitted as a zip archive in the INGInious *Assignment 3 - Avalam: Contest Agent* task.
- **All the files required for your agent to work shall be in this archive, including provided files (e.g., `avalam.py`, `minimax.py`, etc.)**.
- Only agents being able to defeat or obtain a draw from the dummy alpha-beta player in the INGInious *Assignment 3 - Avalam: 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 parallelization your algorithms.
- Your agent cannot use a too large amount of memory. The maximal limit is set to 2 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 `play` method.

Competition rules

The contest will be played with the Avalam rules for 2 players described in the Avalam Instructions file provided on the Moodle course page. Each agent will get a time credit of 20 minutes per match.

There are 42 groups, thus hopefully 42 different agents. These will be divided into 7 pools of 6 agents. 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. If a draw occurs, both players get 1 point. 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. To evaluate the speed of a player, we divide the total time he played during his matches and we divide it by the number of actions he performed (i.e. the speed of a player is the average time he takes to play a move). 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.

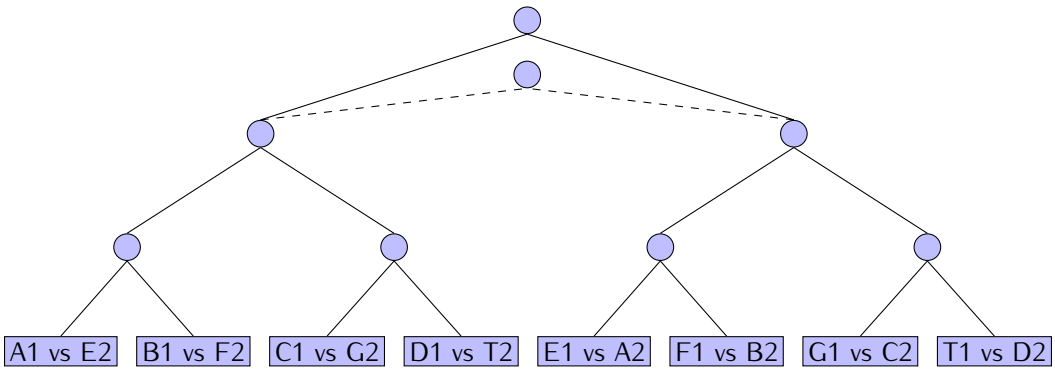

Figure 4: Playoffs

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/avalam2015`. If you want, you may also give your agent a nice name.

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 into great ideas.

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