Report LINGI2261: Assignment 3

Group N°29

Student1: Ben Haddou Mehdi

Student2: Hennebo Eliot

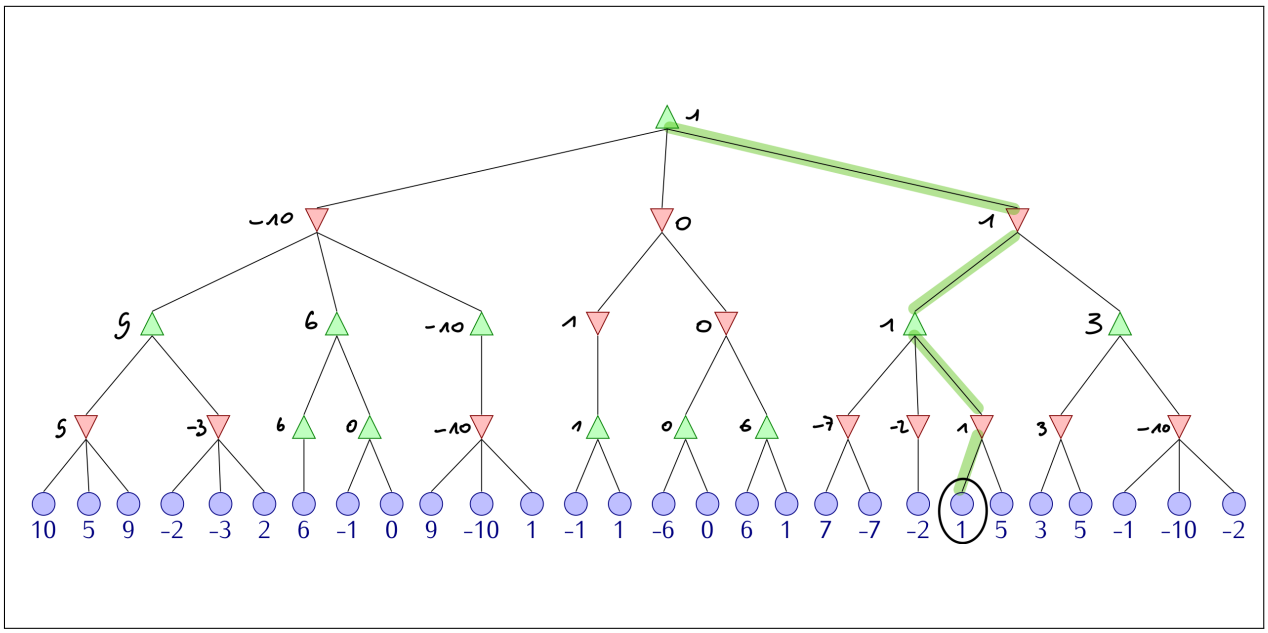
April 22, 2021

Answer to the questions by by adding text in the boxes. Do not modify anything else in the template. The size of the boxes indicate the place you *can* use, but you *need not* use all of it (it is not an indication of any expected length of answer). Be as concise as possible! A good short answer is better than a lot of nonsense ;-)

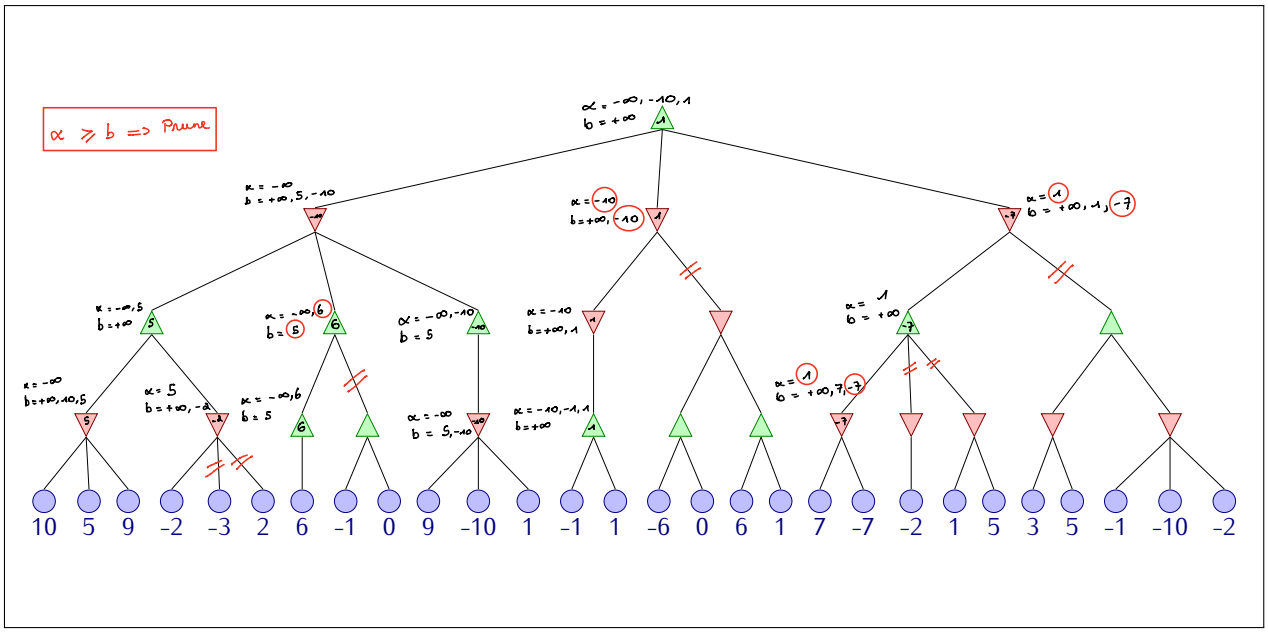
1 Alpha-Beta search (5 pts)

Answer questions 1 to 4 by inserting an image of the tree in the corresponding box. This may either be an edited version of `minimax.pdf` or `minimax.png` (using paint, gimp, etc.), or a photograph of a drawing you made by hand. In any case, the images must be **clear** in order to be graded.

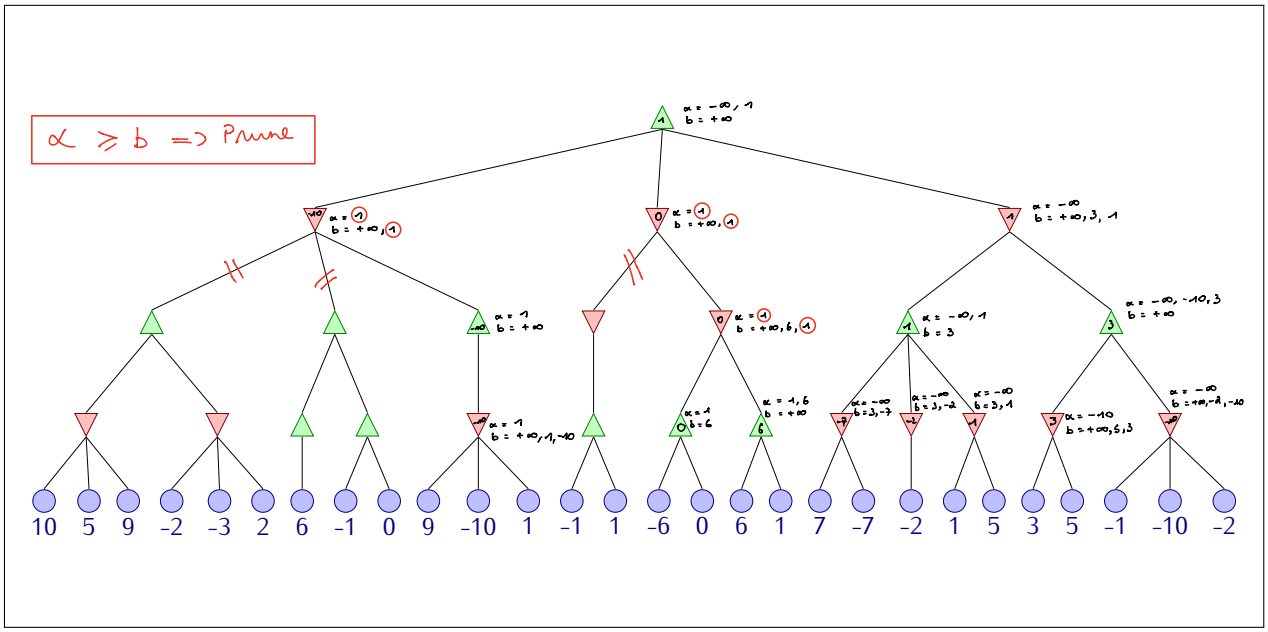
1. Perform the MiniMax algorithm on the following tree, i.e. put a value to each node. Circle the move the root player should do. **(1 pt)**



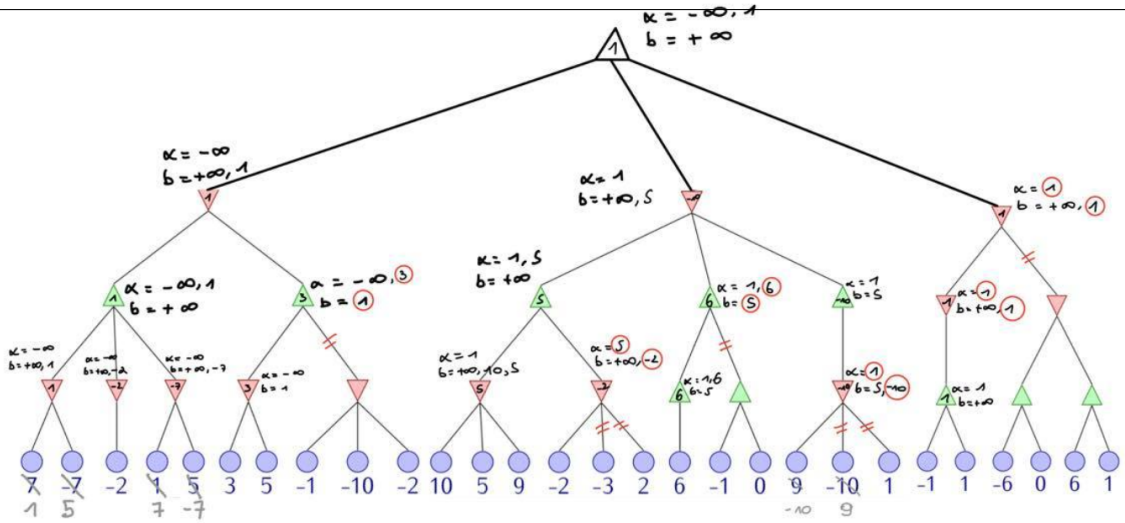
2. Perform the Alpha-Beta algorithm on the same tree. At each non terminal node, put the successive values of α and β. Cross out the arcs reaching non visited nodes. Assume a left-to-right node expansion. **(1 pt)**



3. Do the same, assuming a right-to-left node expansion instead. **(1 pt)**



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. **(1 pt)**



Yes, when the best move occurs at the left side of the tree for each subtree in the whole tree, then it corresponds to the best case of Alpha Beta Pruning.

5. How does Alpha-Beta need to be modified for games with more than two players? **(1 pt)**

First, we need to replace the single value for each node with a vector of values.
For terminal states, this vector gives the utility of the state from each player's viewpoint. The simplest way to implement this is to have a utility function return a vector of utilities.
For non terminal states, the backed-up value of a node n is always the utility vector of the successor state with the highest value for the player choosing at n.

2 Seega (35 pts)

2.1 A Basic Alpha-Beta Agent (5 pts on INGInious; nothing to report)

2.2 Evaluation function (5 pts)

5. What are the weak points of the evaluation functions of your basic agent? **(2 pts)**

1. It calculates only on the basis of the score of our player at a specific state. So it lacks the ability to penalize the opponent's score.
2. The evaluate function does not differentiate between the two phases (placement and game) and therefore plays randomly for the placement phase although it is as important as the game phase.
3. The function does not penalize boring moves and can therefore end up repeating unnecessary boring moves.
4. The function does not use the time allocated to it as the complexity of the function is O(1).
5. The function does not take into account the opponent's score.
6. The function does not take into account the subtleties (or just the obvious things) of the current position that give advantages or disadvantages. For example, the fact that the middle square cannot be attacked, that the corners are more difficult to attack as well as the borders.

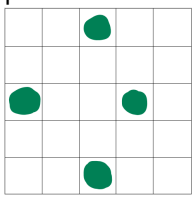
6. As described in the class, an evaluation function is often a linear combination of some features. Describe new features of a state that can be interesting when evaluating a state of this game. **(2 pts)**

Indeed, an evaluation function is a linear combination of differents features which reward or penalize in function that they are an advantage or an inconvenient. This reward or penalization is multplicated by a coefficient in function of his impact.

Before describing some features, it's important to separate the two phases of the game which have different strategies ;

A) Placing phase :

1. It's strong to have a pawn around the middle and even stronger if you have a pawn placed in the border middle at his left, right or in front of him because it allows you to take pawns at the first move.



2. It's strong to have pawn in corners and at borders.
3. When your are the black player it's bad to have pawns at the middle between two opposite pawns.

B) Game phase :

1. It's strong to have a pawn in the middle cell because he his invulnerable.
2. It's strong to have position where one of your pawn can take others pawns next turn.
3. It's strong to form a barrier when you have an advantage.
4. It's strong to have pawns in corners and in borders.
5. It's bad to have pawns blocked by a barrier.
6. It's bad to have pawns which can be taken next turn.

7. Describe precisely your improved evaluation function. **(1 pt)**

Our improved evaluation function is a combination of some features described above with a coefficient in function of the feature impact. As described above, it's important to separate the two phases of the game ;

A) Placing phase :

1. We reward highly (+3) a position where a pawn his around the middle (with a limit of 2 pawn around the middle maximum)
2. We reward highly (+3) a position where a middle pawn can work with a pawn in front of him or at his left/right. (cfr. 6.A.1)
3. We reward moderately (+2) a position where a pawn his in a corner or in a border.
4. When you are black we reward moderately (+2) when a black pawn at the middle is protected by another black pawn behind him.

B) Game phase :

1. We reward highly the difference between the pawns you win and lost. (won + (MAX_SCORE - lost))*10
2. We reward moderately (+2) a position where a pawn is at the center cell.
3. We reward low (+1) when a pawn for each adjacent opponent's pawn, it's favours the creation of barriers and blocking of opponent's pawn.
4. We reward highly (+3) for each pawn you can take in one move while paying with conflict induced by next point (B.5.). For example, case where an opponent pawn can take in one next turn but you can take this pawn in one next turn.
5. We penalize highly (-3) for each pawn you can loose in one move.

2.3 Successors function (4 pts)

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. **(1 pt)**

The branching factor differs depending on whether you are in phase 1 or 2.

In phase 1: It is easy to see that the number of successors starts at 25 and decreases by 1 for each move until 1 last legal move is left.

In phase 2: The number of successors is also low at the beginning of the game, due to the large number of pieces on the board, and increases as the pieces are captured. The number of successors is intuitively higher when there are not too many pieces on the board. This makes the calculation difficult, but by averaging over several games, one can estimate that the cumulative branching factor for both phases is around 8 to 12. However, the maximum can be much higher than that and in a very open situation it could be closer to 30.

9. What do you loose by ignoring some of the successors? **(1 pt)**

That approach is called Beam Search, we consider only a "beam" of the n best moves (according to the evaluation function) rathen than considering all possible moves. That approach is dangerous because there is no guarantee that the best move will not be pruned away.

In our game, this could mean ignoring a node because its evaluate function is disadvantageous for the next move, to draw a parallel with chess, a trap can often start from a move that seems bad at first, but which can cause an even bigger mistake from the opponent. This kind of move would be ignored here if we rely only on the evaluate function because it does not consider it a trap. Very concretely, a sacrifice move that would win two pawns the next move is ignored because it is considered by the evaluate function as a move that loses a pawn and can therefore be in the % of worst successors.

That approach is clearly different from the pruning in the Alpha Beta Pruning algorithm, as it has been proven that in the Alpha Beta, the pruning has no effect on the result, and in this case we lose nothing.

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? **(1 pt)**

The effectiveness of alpha-beta pruning is highly dependent on the order in which the states are examined. The fact that we can effectively prune some branches reduces the number of explored nodes from $O(b^m)$ in Minimax to $O(b^{m/2})$ in Alpha Beta Pruning, meaning that the branching factor becomes $\sqrt{b}$ instead of b. The alpha beta can then solve a tree roughly twice as deep as minimax in the same amount of time.

This is great but it is the best case and it of course depends on the ordering of the tree, if the tree is ordered with the worst case, the complexity of Alpha Beta Pruning is equivalent to Minimax.

There are different ways of ordering the tree to help Alpha Beta Pruning doing a good work. In seega it can be smart to order the successors already on the basis of the moves in which you can win a pawn, this joins the killer move heuristic, beyond this ordering, you can apply an order on the basis of the evaluate function, which will by definition rank the most interesting moves according to our evaluate function.

11. Describe you successor function. **(1 pt)**

First of all, we look at which players we want to generate successors for. If it's our player, we generate the successors and sort them in ascending order of the evaluate function to help pruning. If it's the opponent we sort them in descending order to help the pruning. Also, we check if the move that is generated blocks the opponent, if it blocks the opponent we pass it, in order to remain fair play and respect the rules of the game.

Finally, if the number of successors is >6, we remove 50% of the successors in order not to overload the alpha beta, this follows the forward pruning approach which consists in removing part of the least interesting successors on paper, this method has disadvantages as explained later but it still allows to analyse interesting moves very deeply which can make the difference in a match.

2.4 Cut-off function (4 pts)

12. The cutoff method receives an argument called depth. Explain precisely what is called the *depth* in the minimax.py implementation. **(1 pt)**

The depth in the minimax simply represents the level of the tree to which the algorithm is applied, each level represents a move, from one player or the other, and there can be moves from different players on the same level since the players can play several moves in a row.

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 50 boring moves have occurred. Why could such an approach be problematic for an Alpha-Beta agent? How could you remedy this? **(1 pt)**

If a barrier is built, it means that the Alpha Beta agent has come to the conclusion that this is the most logical sequence of best moves. This implies that there is no particular reason to get out of this situation, because getting out of this situation would certainly involve making a disadvantageous move that would unblock the situation. This is not intuitive for the Alpha Beta agent who does not logically expect to make a disadvantageous move just to continue the game normally and it is thus a problem.

This can be seen in two ways:

1. If we are winning and a barrier is created, it is to our advantage to make 50 boring moves, so that we can win thanks to the rule of who has the most pieces at the end in case of a draw.
2. If we are losing, on the other hand, it is absolutely necessary to avoid making boring moves, so as not to lose because of the rule. In this case, it is necessary to specify to the algorithm that it will have to make a move to unblock the situation, and that the optimal move is not the one which maintains the barrier. To specify this to the algorithm, we can simply apply a malus in the evaluate function to the boring moves, this will have the effect of bringing up moves which are a little less interesting on paper but which will unlock the situation

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. **(1 pt)**

We set a time limit per move that is equal to a percentage of the remaining time. So that at the beginning of the game, when the moves are important and have consequences because the game is still very close, it has more time to look for the optimal move. His time per move decreases as the game goes on, which is intuitive if we assume that the game is less and less tight the more we advance, and that optimal moves are no longer essential, for example if we have a big advantage.

The fact that we use a percentage of the total time available makes the possibility of time out virtually impossible because the algorithm will automatically speed up as time progresses. It also automatically adapts the agent's level according to the time of the game, he will automatically play better in a game with more time, because he will allow himself more time to explore more nodes.

Iterative deepening allows repeating the alpha beta with limited depths and incrementing these depths as you go. You can therefore have a variable depth depending on the time available. As the time given to a move is determined by a percentage of the player's total time, we can examine all the moves up to a certain depth and increment if there is time left. This solves a major problem that arises without Iterative Deepening, because if the defined depth is a bit too ambitious, and not all available moves up to that depth have time to be examined at the time limit, an optimal move may be forgotten due to lack of time.

15. Describe your cut-off function. **(1 pt)**

Our function determines whether the time to search for a move has expired and the depth in the tree to which it is tolerated to search

1. For phase 1: we establish a time per turn and depth limit.
2. For phase 2: we establish a time per turn and depth limit but it is higher than the one in phase 1. The time per turn is calculated by a percentage of the total time left for the player. (1.5% in the phase 1 and 3% in the phase 2).

Then we measure the time elapsed since the beginning of the player's and we evalue the three conditions to return our boolean value :

1. The time has not elapsed
2. The maximum depth has not been exceeded
3. The game is not over

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

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

18. Describe concisely your super tough agent. **(5 pts)**

First, we use several parameters:

1. phase1time = 0.01, which allocates time to phase 1 moves.
2. phase2time = 0.025, which allocates time to phase 2 moves.
3. phase1maxdepth = 12
4. phase2maxdepth = 10

These values have been arbitrarily chosen via numerous tests according to the most efficient time/depth ratio.
Secondly, we use iterative deepening combined with the minimax algorithm to know which move we will make in the next round. The iterative deepening will run the minimax algorithm with a depth from 1 to N and keep the best move proposed. We allocate 2.5% of the total time for each move or we stop when the depth 12 or 10 has been reached.

The cutoff function is used to know if the time has been exceeded, if the current depth has been exceeded or if the game is over.

The successors function generates the possible moves of the next player. First of all, the discard function generates the moves that block the opponent if he is already blocked.

Secondly, it discards moves that have already been encountered in the generation of successors for the current move. It also sorts the moves according to the function evaluate in decreasing order when it's our successors and in increasing order when it's opponent move. If the number of successors is greater than 6, we manually prune 50% of the worst successors.

Finally, the most complex function is evaluate. It is divided into two phases;

The placing phase where the goal is to optimize the placement in order to start with an advantage that is not directly written in the score. In order to optimize the placement, we give rewards when the player puts pawns that will be able to take opposing pawns in the first round. We also avoid being caught by this strategy when playing black, so we give rewards when a black pawn in the middle is protected by another black pawn. We also favour the plating of pawns on the edges and in the corners.

For the second phase, we give rewards when the player sticks the opponent with his pawns as well as when he puts pawns in the midd