## 

To define the Successor function, we first define the Current location of the agent 1, and agent 2, as following:

Since the state variable doesn’t contain exact information on the agent 1 and agent 2 locations, we have to loop through and find a tile with values 1 and 2.

We have to check that the following conditions hold:

* Player can move to a tile which isn’t gray, or other player
* Player can move to a tile which isn’t a wall

Defining a group of Moves which can be done in a game:

For example, a move of first player upwards is: , where:

Defining a condition which will be checked:

Now, we can define the successor function:

## 

Win(s,i) ⬄ Succ(s,i ) ≠ ∅ ∧ ∀ s’ ∈ Succ(s,i): Succ(s’,(i+1)mod 2) = ∅

Tie(s) ⬄ Succ(s,1 ) = ∅ ∧ Succ(s,2 ) = ∅

## 

The branching factor can be 4 in the initial turn of a any player, if surrounded by unvisited or white tiles. From second turn on, it will be 0-3, since we can never visit the tile which we have already visited.

## 

Simple player iterated on all the directions (4 directions – UP, DOWN, LEFT, RIGHT), checking if they are feasible. If a direction is feasible, the ‘state\_score’ is being calculated. The move with the best score is performed. If state\_scores are equal, FIRST direction which was calculated wins.

According to the ‘state\_score’, it will prefer the state with the LEAST number of further states available for that player, as long as this will not lead to 0.

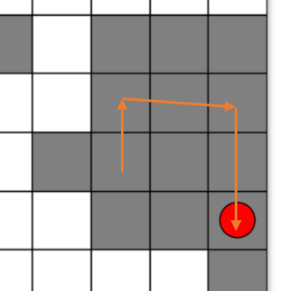
PROS:

* Given a certain area with no obstacles (a large space), the agent will for sure fill all the tiles

CONS:

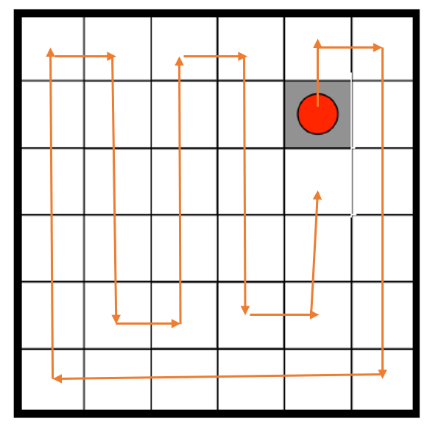
* Sees only 1 step into the future
* Doesn’t take enemy location into account
* Can run into the dead end, while being able to easily avoid it (connects to the first CON)

Example of a dead-end (map number 3)



## 

As stated in PRO, the agent will fill the whole given area in an optimal way, for example here, I have marked it’s path:



## 

The heuristic value includes only the number of successors of a certain state for a certain player. CONs can be similar to the “simple player” tactics, and more:

* This heuristic is not admissible – Sometimes going into a child state which has only 1 successor can lead to victory, while still having other children with more successors, which will result in higher heuristic
* It does not take into account the enemy location
* It looks only 1 step further.

## 

First let’s define 3 components:

1)

AvailableTiles(S) = white\_tiles/total\_tiles. This component offers us an insight on how advanced the game is, the bigger this component is, means we are at a later stage in the game.

2)

GroundDiff(S) = Player\_ground – Opponent\_ground. Player’s ground is defined by the sum of the locations that are available, and closer to the player than to the opponent( in terms of number of moves). The same goes for the opponent’s ground.

3)

Opponent\_dist(S) = the number of moves it will take for the player to reach the opponent.

Given these components we can define a heuristic as follows:

H(S) = AvailableTiles(S)\*GroundDiff(S) - (1-AvailableTiles(S))\*(Opponent\_dist(S)

This approach will be consisting of two stages: In the beginning of the game, meaning when there are more white tiles, the player will try to collect a big territorial advantage on the opponent trying to maximize his ground and minimize the opponent’s ground.

As the game proceeds and the white tiles go and diminish, the player will take an increasingly aggressive approach by trying to get closer to the opponent in order to close him.

These specific components were chosen as they give as close as a full picture of the state of the game as possible, providing crucial information in order to make the next move.

## 

The “anytime” variation of the Minimax algorithm consist in running the usual minimax algorithm with an additional time parameter, and demanding that the algorithm returns the best answer available in that time.

For this purpose we run the rb-minimax algorithm with increasing depths, until the available time runs out.

At the end of each iteration the chosen step is saved and at the end of the allocated time we interrupt the search and return the last chosen step.

This technique is called iterative deepening, and the problem presented in the lecture regarding this technique, is that on average, the time will run out during the last iteration calculation, thus we will return the previous iteration step, but still most of the resources will be used by the last iteration, so we will use a lot of resources, for no gain since we will not manage to terminate the last iteration.

## 

The proposed solution for this problem in the lecture is that in each iteration, we can store the minimax value for each of the sons of the upper level

On the last iteration, thus, half of the sons will represent a search at the deepest level, whereas the rest will represent the search at the previous depth level ( one before the deepest). This will solve the problem partially, since at least for half of the sons we will have used our resources in order to see into the next depth level.

## 

Defining the function:

Looking at the Minimax basic agent (or Alpha-Beta with NO PRUNING (worst case possible, even with child sorting)), the next depth will develop leaves, where is a branching factor.

Define:

We know, that in BFS,

Thus,

Defining number of children developed at current depth as , we get

Assuming that the time to develop each leave is the same among every iteration, we get:

We define the time to develop leaves at the depth as

We take the maximum branching factor possible (after the first move), which is .

Thus, we get:

We also need to account for the time required to develop all the previous tree again, thus adding to the total time for the calculation of the whole tree at level :

This is of course the worst case, where no pruning occurs, and that EVERY leaf has 3 successors.

*Note:*

In any case, the actual time may be lower because the worst case for the branching factor was taken.

When using pruning, the actual time required for the next step maybe even less, since some of the branches may be pruned, thus we will save time on their calculation time.

## 

In the Anytime Contract, the Alpha-Beta variation is supposed to be more efficient. We have to remember that it prunes only the branches that will anyway not affect the decision of the root node. Thus, those resources (more time that became available) will be used to reach deeper depth of the tree.

## 

When the depth will be limited, both algorithms will return the exact same result. As mentioned before, Alpha-Beta prunes only branches which won’t affect the Minimum or Maximum choice of the parent node.

## 

The heuristic for the Minimax player is the same heuristic as defined in the Question 7 in this report. The scale factors were chosen statistically and experimentally.

## 

As described earlier, the heuristic is based on 3 main parts:

EQUATION

* When the distance to the enemy is large (perhaps in the beginning of the game), the agent will go in the direction of minimizing this distance.
* When the distance is smaller than 10 (value chosen experimentally), it will not affect the heuristics.
* Another heuristic value, which will be more dominant when the agent comes close to the enemy (the distance heuristic will be already small) is the MaxGround heuristic. The agent will calculate the tiles advantage of a player against the opponent.
* Also, we have a third heuristic value, StateScore, which calculated the number of free tiles around an agent.
* We also use the Utility function. If some state is found as “final”, it calculates who wins after this state. If player wins, a value of “99999” is returned, and “-99999” if an opponent wins. The value of “-9999” is returned if the result is a tie, so that the agent will prefer choosing another choice which is not final, and has some heuristic value returned.

## 

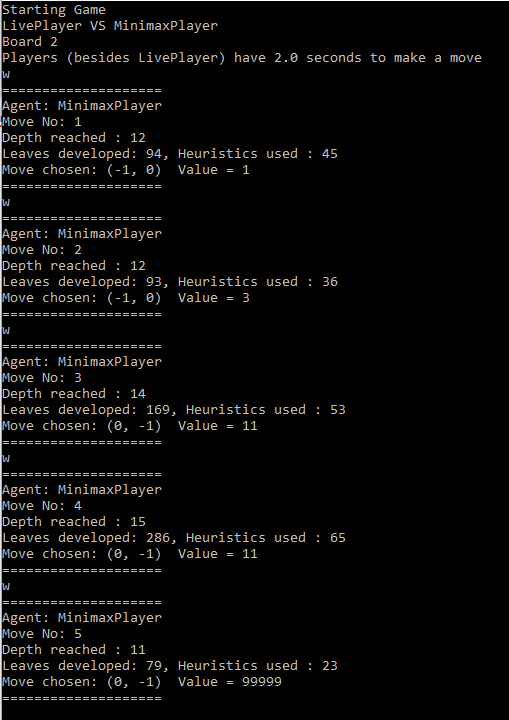
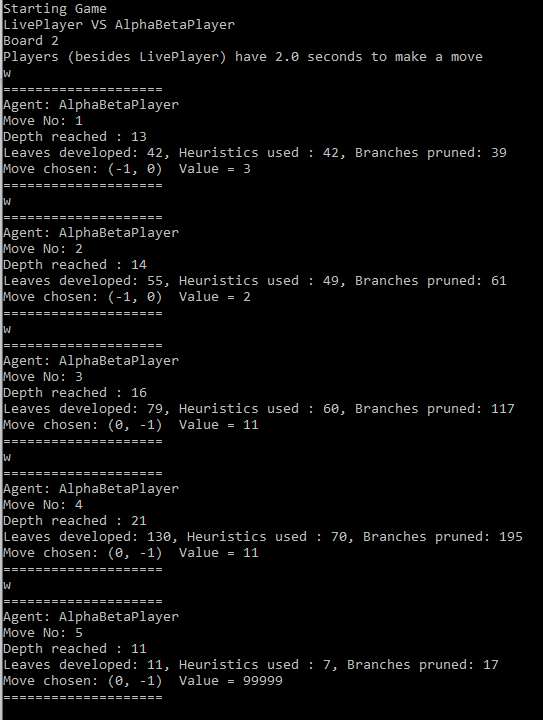
The theoretical explanation was given in Question 10. The implementation was made as described in the homework assignment paper, since it was found to be efficient, and indeed there were no times when the time limit was broken.

## 

The expected behavior : the Alpha-Beta Player is expected to reach further depth in a given amount of time than the Minimax Player, thus giving a better estimated Minimax value and its corresponding move.

We can set players to play one against the other, or rather we can play against both of them, performing same actions, which is what we do. This way they will have same starting conditions. Those are our custom outputs from the game environment.

The Minimax player is on the left and the AlphaBeta player is on the right.



“Value” symbolizes the Minimax value returned for this move. When this value is 99999, this means the agent has reached the Utility function which returns the “Victory” for the agent, thus there is no need to develop further depth, and it stops and returns the action which returned “Victory”.

We can see that AlphaBeta Player reaches deeper upon every step calculation.

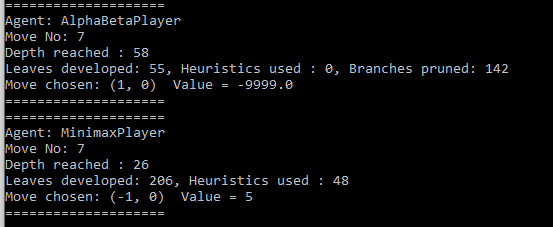
We can clearly see that the AlphaBeta player prunes branches (it grows from turn to turn because the number of free tiles in the game diminishes, and heuristics calculation is made easier each consecutive turn), which allows him to reach bigger depth and prune more branches.

We can see that Minimax develops more “Leaves” (Nodes where heuristic or utility is calculated), which shows that it doesn’t prune any branches and develops them even though they don’t affect the Minimax value at the root.

Everything is as expected, and other boards show similar behavior.

We also let the agents play against each other, letting each agent to start first or second.

We can see also in game that the AlphaBeta agent sees further, and reaches bigger depth, thus can see the outcome of the game before other agents. For example, here it calculated that the game will end with “Tie”, while the rival (Minimax) still used heuristics. This happened on multiple boards.



On the small boards there is a bigger chance of a game ending with a tie, since the heuristics give a good evaluation of the movement until the moment when the amount of free tiles is relatively small, so that both agents can calculate the whole tree until the “victory”, “loss”, or “tie” outcome.

Those are the results of the games with Same heuristics for both players:

|  |  |  |
| --- | --- | --- |
| Board | Starting Agent | Result |
| 0 | Minimax | Tie |
| 0 | AlphaBeta | Tie |
| 1 | Minimax | Tie |
| 1 | AlphaBeta | Tie |
| 2 | Minimax | Tie |
| 2 | AlphaBeta | Tie |
| 3 | Minimax | Tie |
| 3 | AlphaBeta | Tie |
| 4 | Minimax | AlphaBeta wins |
| 4 | AlphaBeta | AlphaBeta wins |

We can see that the difference in outcome starts being visible on big boards.

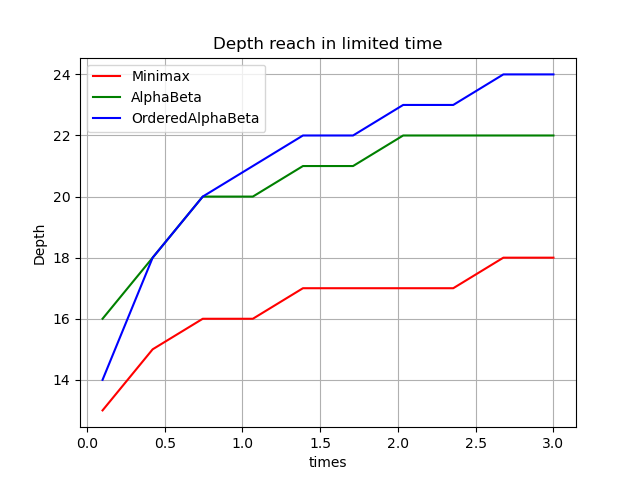
## 

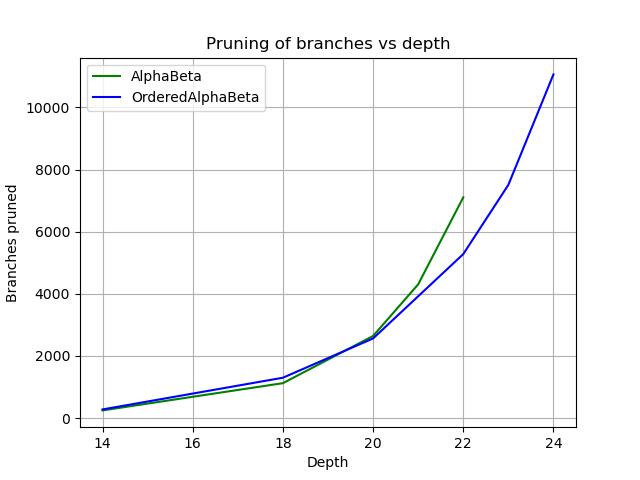
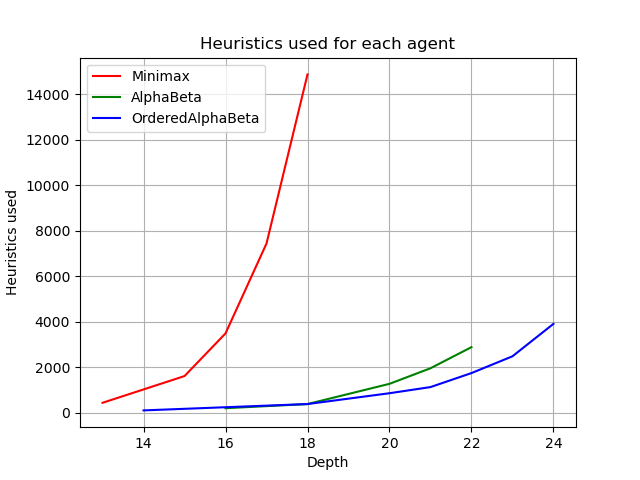
The example code was used and extended.

Before running the graphs, our expectations are:

* Given a certain amount of **time**, Ordered Alpha-Beta would reach biggest depth, then regular Alpha-Beta. Minimax would reach the most shallow depth.
* For a given **depth**, the Minimax Agent will use the most amount of heuristics, since it does not prune any branches, and has to develop the whole tree.
* For a given **depth**, Ordered Alpha-Beta will develop less branches than the regular Alpha-Beta, since branches will be cut closer to the root, thus their children branches will never be discovered.
* For a given **depth**, The Ordered Alpha Beta will prune less branches than regular Alpha-Beta, out of the reasons states in the previous statement

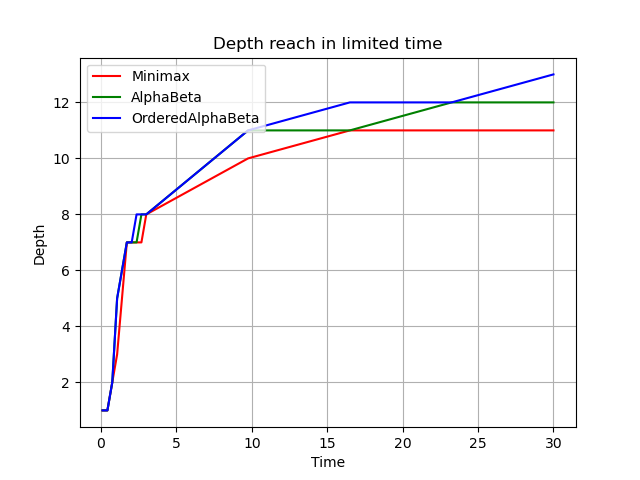
To verify those statements, we do the experiments. First, on more simple heuristic, consisting only of the StateScore value:



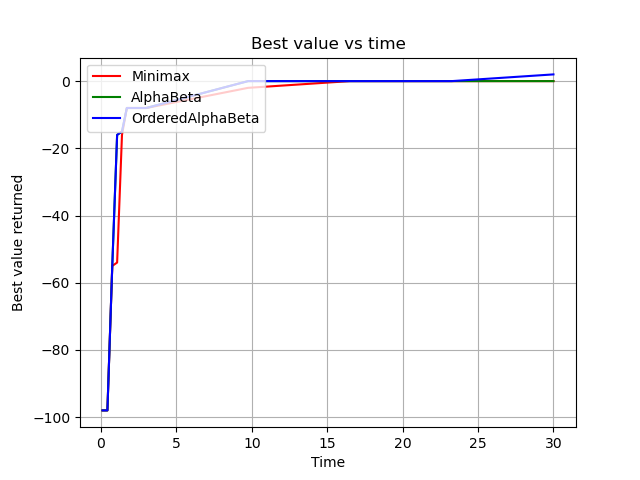


We can indeed see that all those statements are verified on the simple heuristic.

Now, running on the more complex heuristic:



We can observe the same tendency – the Ordered AlphaBeta player reaches the furthest depth, while the Minimax player reaches the shallowest. Another interesting graph to look at is the Minimax value returned. We can see that Ordered AlphaBeta succeeds to return the biggest value for a given time, thus providing us with the most beneficial (according to heuristics) move.



## 

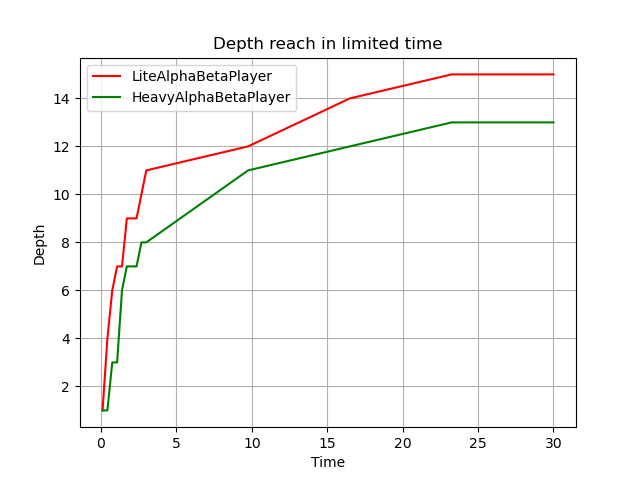
Two agents were created

The difference in the Heuristics was only the MaxDistance parameter for the MaxGround calculation. To remind, this heuristic calculates the difference between the amount of free tiles that are closer to the agent and the amount which is closer to the enemy. Only tiles below the MaxDistance are taken into account. The other 2 heuristics are light already, so they are left equal between the Agents.

What we expect:

* The Depth reached by the Lite player should be higher, since the Heuristics calculation time is smaller

And indeed:



We can see that this is the case.