Player documentation of "Loisel"

We started the whole issue by implementing a player that uses the minimax strategy. Because of the fact that this player investigated many useless nodes and therefore was very slow, we extended it to alpha-beta pruning to be able to search deeper and faster in the tree.
Still, in many games the game tree is even too large for this strategy, so that we had to think of some heuristics to improve the search even more.

The main problem of every standard strategy is that it is only useful when terminal states can be reached. However, this is rarely the case, at least at the beginning of the game. Due to this, we developed an evaluation function, exploiting the degree of freedom of the opponent as a guide. This means, that we check the number of legal moves of the opponent and try to minimize it.
As an example, consider the game chess, where minimizing the opponent's possible moves leads to reducing the number of his men on the board or to forcing a check. Here we can see that minimizing the number of moves of the opponent can be a good idea.

First, we based this heuristic on a depth limited function, so that alpha-beta-pruning was called with a fixed depth limit.
However, the size of the game tree varies from game to game (tic tac toe vs. chess), so that we thought of an approach with more flexibility.
We then ended up with Iterative Deepening. In our case this means that we start with the cutoff value 1 (i.e. layer 1) and while there is still enough time, we increase this value by two after every step. This is because we are only interested in every second layer of the tree, because that is the layer in which the opponent moves. Based on the amount of legal moves we then can apply our evaluation function.

Additionally, we modified the alpha-beta-pruning strategy slightly so that it already makes the cutoff when a node with utility 100 has been found. This makes sense because in the games the players are playing 100 is the maximum outcome. Another reason is that the goal state we found is the most closed one to the start state because iterative deepening uses a breadth-first-search strategy. A goal state close to the start-state is always the best, because the other player has the fewest possibilities to change the game to another direction.

After alpha-beta-pruning has finished and returned a list of possible moves, we check whether this list is empty, which may be the case due to time constraints. If it is empty, we just take the move list generated by the previous iteration.
In the following step we sort this list and take the last, that is best move.

Since our evaluation function never returns a value higher than a winning move, we always take a winning move, if one exists.

As a conlusion, here is an informal description of the relevant steps of the algorithm:

```
deepness = -1
WHILE (enough time)
read previous move list
new move list = alpha-beta_pruning(..., Deepness + 2, ...)
if (tree smaller than deepness or winning move found)
break
else
continue
END WHILE

if (new move list is empty)
new move list = previous move list
sort move list

take last element
```

In the end we tested the player with various games and opponents. The results show that it always plays optimal in small games like tic tac toe. Interestingly, it has very high skills in playing chess, which probably is based on reasons already stated above.
On the other hand, a major disadvantage of our player is that it does not take into account the difference between the number of legal moves. That means it just looks how much legal moves the opponent has but does not care about how much he has himself. This mostly leads to an aggressive play that is highly decision-oriented.
Another basic problem of algortihms like alph-beta-pruning is that it is only useful for games with alternating moves.