Selected Optimization Methods

final project documentation

Damian Woźniak

Computer Science (sem. V, extramural studies)

1 Theoretical description.

The purpose of this project is to present an algorithm that can be used to find an approximation of an optimal move in a game. It is presented on a specific game, but the general idea applies to any turn based game with any number of players.

1.1 Building search tree.

In mathematics tree is a type of graph that has no cycles. We are going to use a tree to create a model of possible game states. Each node of the tree is going to represent game state. Root represents current situation, first level represents states after each possible move for the current player, second level represent states after moves of the second player. We can observe that number of nodes grows exponentially.

In a game with two players where each player gets to make only one move per turn nodes at odd levels represent game state after moves of one player and nodes at even levels represent game state after moves of the other player. Leaves represent game states where the game ended.

For many games creating full free is impossible. It might contain infinite branches where game state changes back and forth, or it might be just simply too big to compute. As we noted before, number of nodes grows exponentially with each move.

Simplest heuristic is to cut the tree at a given level. However doing so often leaves us with a tree with branches that are all the same size, and gives us no basis to choose one move over another. To compensate for that we can add value to each node.

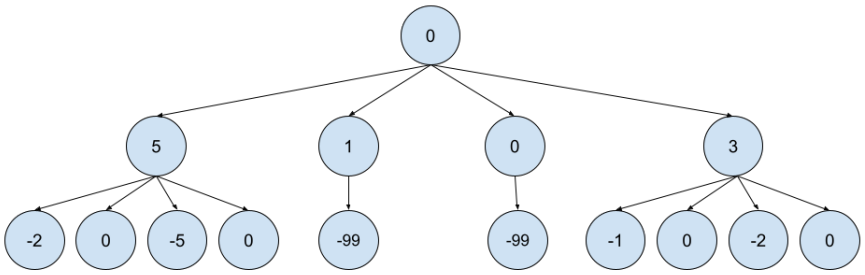
In our example we use traditional formula of evaluating move based on the value of a piece that was lost. Pawns are worth 1 point, knights and bishops 3 points, rook 5 points and queen 9 points. If the current player takes opposing piece points are positive, if it is the opponent that takes piece of the current player the assigned points are negative. I also decided to encourage AI to be more aggressive by adding small bonus to moves that result in check and to moves that put pawns on the enemy territory because getting your pawn to the end of the board is often a game winning move. Moves that end the game have 100 and -100 points assigned to them because if you have opportunity to make such move or prevent opponent making such a move it is the most important thing worth any sacrifice.

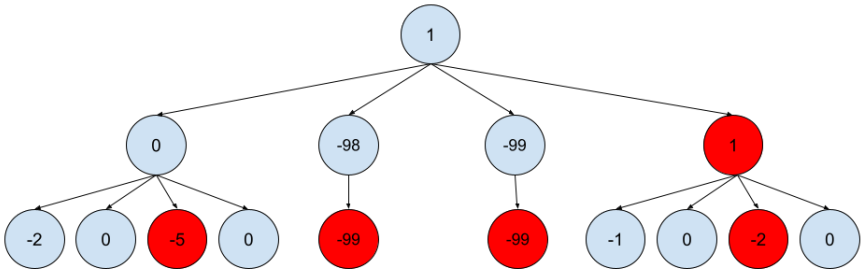
1.2 Searching

To search the tree we could use one of the recursion algorithms. Value on each node represents how good or bad the situation in the game is for the current player. Starting from the lowest level we add value of the child node with the highest possible value if it is the move for current player, or value of the lowest child node if it is move of the opponent (this way we assume our opponent will always make move with the highest value, which is not always the best move but it is a good approximation). After doing that for the lowest level we continue to do that for second to last and so on. We stop at first level after root and choose the node with the highest value. That node represents move that was chosen by our algorithm as the best of all possible moves.

2 Exemplary calculations

Below i present an example of how a simple two level search tree might look like, and how nodes are chosen. The red ones are the chosen as the best moves for that player. So at second level we choose nodes with the lowest value because that nodes represent moves of our opponent. Then we add the chosen value to the parent node, and choose the node on that higher level this time selecting node with the highest value.





3 Supporting computer program

3.1 Github repository with whole project.

`https://github.com/wutek/chess`

3.2 Demo version where you can play online.

`https://wutek.github.io/chess/`

Please note that the program uses service workers with modules that are not yet supported in every browser. Please use Chrome or Opera browsers that are currently able to handle that.

4 Room for improvement

There are few things that could be added to the project to make the algorithm better.

In the beginning of the game there is little to no opportunities of taking opposing pieces. That leads to all moves being evaluated at the same value and one of them being chosen basically at random. Solution could be to add more sophisticated value system where we take into account positional advantage - for example number of fields that are covered by potential check of our pieces. Another solution would be to add book of openings that are regarded as the best standard openings and use them for as long as the opponent is moving with the predictable pattern.