ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

PyChess_ADSA

ADSA-4A-IBO-FINAL-PROJECT

Karim MOUADDEL – Victor QUERETTE

05/12/2017

Summary

Introduction

Context

Our ADSA project was about to implement an AI for a game to be picked up in a list given in the subject. The subject allowed us to choose a game that was not on the list provided that we use notions related to trees and graphs to implement our AI. Being chess aficionados, and knowing that most modern chess AI rely on computing potential moves using trees, we did not hesitate on measuring ourselves to the greatest of all games – the Kings' Game. A reader unaware of the rules of chess can find them by following this link : https://en.wikipedia.org/wiki/Chess

Chess is deterministic game – its issue only depends on the behaviour of the two opponents. Chess has not been computationally solved yet : we do not know of any strategy that is guaranteed to win. We do not even know if such a strategy exists, and if such « perfect playing » is possible from two opponents, we have no idea whether the game favorizes one or the other of the « white » and « black » players.

All this uncertainty remains even though nowadays machines allow chess AIs to cruelly outperform the best human players : the current best chess engine, asmFish, hits an Elo score of 3425, while the all-time best rated human player, Magnus Carlsen, peaked at 2882 Elo in may 2004. But there is a long way between beating human players and solving the Game, considering that its game-tree complexity was shown by the American mathematician Claude Shannon to be within order of 10^{120} possible games.

First implementation choices

Our PyChess_ADSA AI, like many chess engines, implements a MinMax algorithm to make its decisions. There are two tricky parts in considering chess with MinMax :

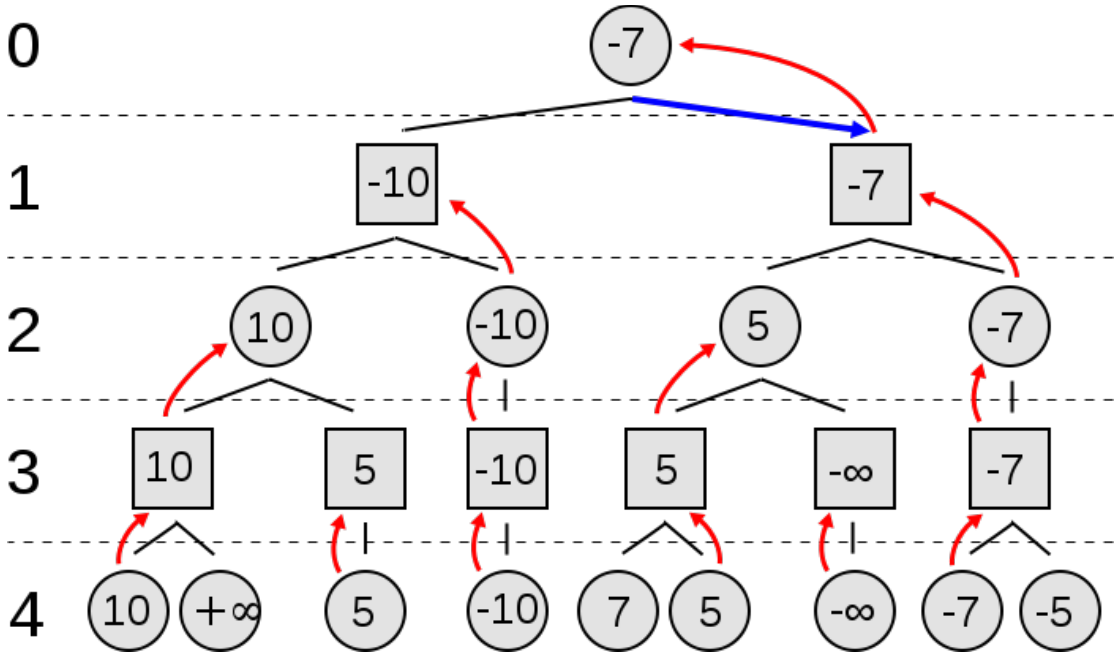
- As stated before, the game tree is deep. Way too deep to be fully generated and parsed in an acceptable space and time complexity. This implies to limit the analysis to a certain depth, and possibly to use pruning algorithms for optimization.
- The evaluation function that attributes values to positions is complex. It is not perfect even with high-end chess engines, for we do not know of all the underlying mechanics of chess.

PyChess_ADSA will be developped without use of any third-party library, excepted the Python SDL-based pygame to provide a proper UI.

Using trees in the design of a chess AI

Trees in chess

The chess engine in PyChess will be designed using the well-known MinMax algorithm. This algorithm, applied to chess positions, aims at minimizing loss in a worst case scenario. The numeric values MinMax needs in order to represent a board will be determined by the evaluation engine, which will be detailed later on.



An example of MinMax

The tree MinMax applies to represents the possible outcomes of the game in the following fashion : every node represents a position, and has for children every possible position directly one move after. Recursively, comlpeting such a tree and parsing it would solve chess.

Levels of the tree are alternatively representing position where either player has to make a move, starting for white at the root of the tree (the untouched, peaceful board).

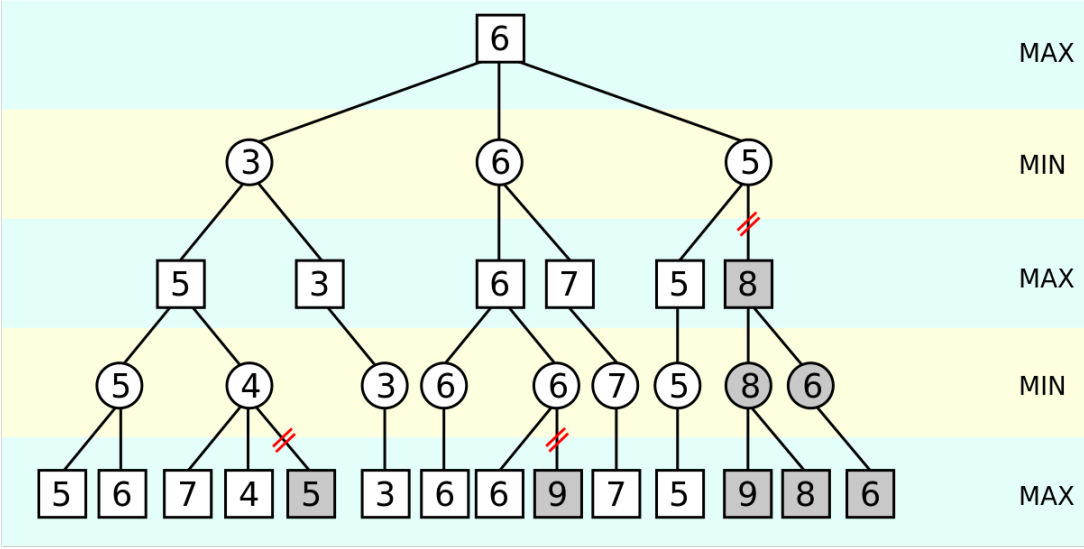
MinMax considers that each turn, the machine's opponent will make a perfect choice, and pick the tree node giving him the highest issue value. Its aim is therefore to minimize this value in advance in order to « minimize loss » and therefore always be in the best possible situation against an almighty opponent.

Optimizing the usage of trees

For every node in the tree to be attributed a value, we have to generate every possible move for the active player in order to get the direct children of the node. Then, recursively and up to a certain depth, we switch to the other player and generate the lower level. The move generation takes little but non neglectable computation.

What takes exponentially more computation, on the other hand, is to evaluate every single position on the lowest generated level of the tree. The poblem is , the immense majority of this computation will be useless because some outcomes, given our depth level, cannot possibly make up in lower levels for their too weak score, and therefore will never be picked no matter their children.

This is where a famous optimization for MinMax decision trees comes in : the Alpha-Beta pruning. By keeping track of alpha (the highest value guaranteed to the maximizer) and beta (the lowest value guaranteed to the minimizer), it is possible to avoid evaluating certain board states that cannot improve the situation for the current player.



Alpha-Beta pruning

Chess heuristics : evaluating positions

Evaluating positions is what makes the quality of a chess engine. Heuristics are not only conceptually difficult : they are computationally demanding. Pertinent choices have to be made on what to measure on the chessboard.

Material

The easiest thing to get an idea of whether a position is good or bad for a given side : who has the more material value (pieces) ? A good model is to count pieces, pondering pawns with a value of 1, knights and bishops with 3 (this can be subject to debate, but at a way higher level, and those coefficients would have to be re-ponderated many times during the game to be accurate…), rooks with 5, and queens with 9.

Pawn structure

As pawn cover a vast space on the chessboard and are prone to be developed early in the game, the structure a player has managed to build means a lot to the quality of a position. Are there any isolated, passed, chained, backward, doubled, tripled pawns ?

Development

Who has more pieces out ?

King safety

Are there any structural weaknesses in either king's position? Does one player have a bunch of attacking pieces near the opponent's king?

All of those parameters, and many more, can be taken into account to design a proper evaluation function. Ideally, this evaluation function would change along the game, adapting its behavior to the opening, mid or endgame…

Conclusion

PyChess is an interesting project that will demand time and effort, but we are motivated by the love of chess and CS !

It is a good opportunity to put to good use what we learned and practiced playing around with data structures in algorithms.

Chess grandmaster Gary Kasparov losing to IBM's Deep Blue in 1997

References

https://github.com/lamesjim/Chess-AI

https://github.com/niklasf/python-chess

https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977

https://en.wikipedia.org/wiki/Immortal_Game quality reference, if any.