MTAT.03.238 Advanced Algorithmics

AI in Chess

Viacheslav Komisarenko, Olha Kaminska and Diana Grygorian

UNIVERSITY OF TARTU
Institute of Computer Science

Task

Our main goal was to create **Artificial Intelligence** based on heuristic search for best move choosing, which could play chess game with user as professional. We wanted to created AI which can beat the weakest well-known analogs.

We used **heuristic search** to choose the best move from all possible in current position on the play board.

Plan of the project

On the Figure 1 illustrated the main parts of the game development.

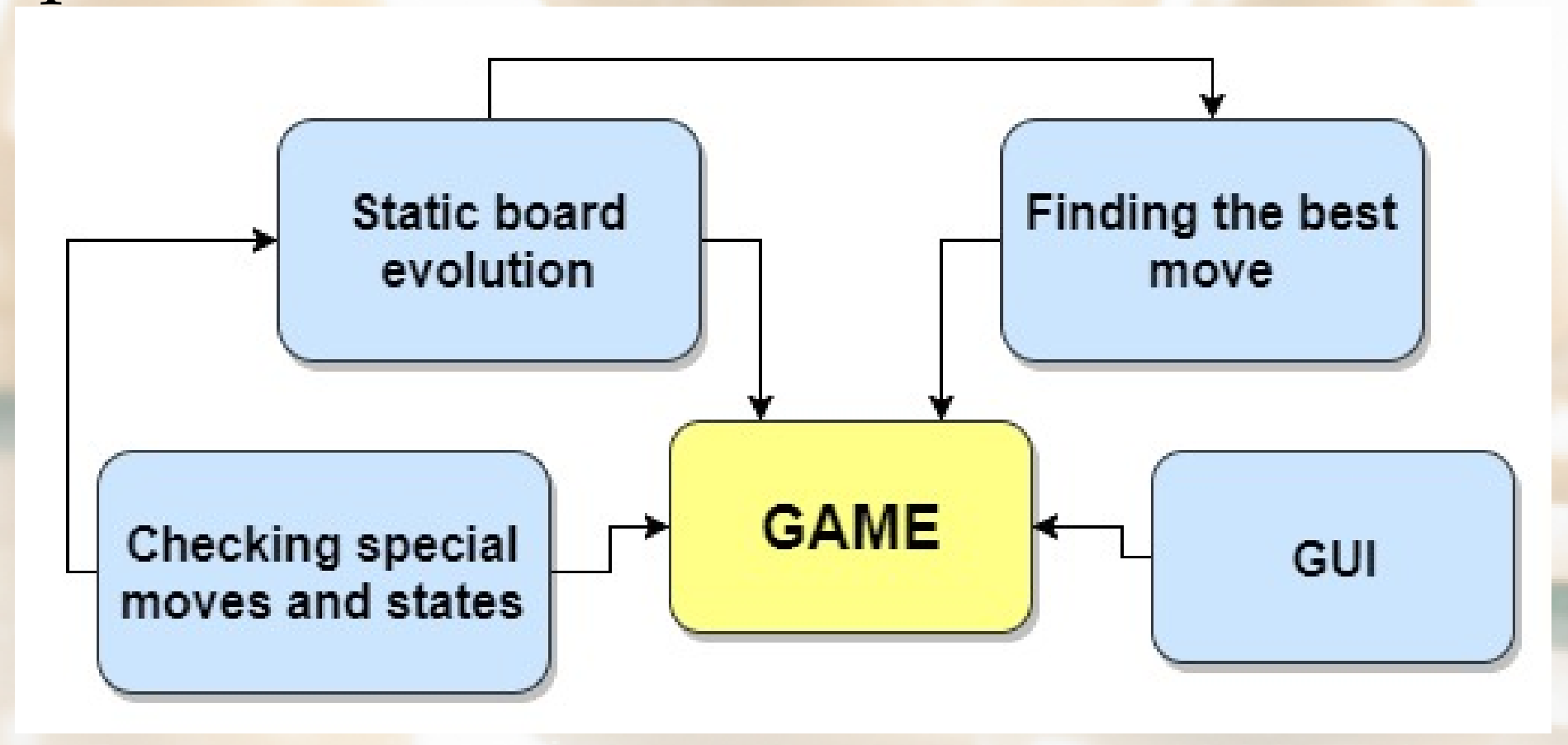


Figure 1: General plan.

General principles

Chess is a two-player, a zero-sum, an abstract strategy board game with perfect information.

The strongest players choose their moves in order to obtain best position in assuming of **best opponent's response**. It can be rewritten in terms of **minimax problem**.

Brute-force approach is not applicable for solving this problem: at each state player can select one move from approximately 20. It means that search for best move on depth more than 3-4 is already hard on computers.

There are methods that allow not to look at each possible variant by pruning inappropriate branches of game tree.

Principal variation search

Principal variation search was chosen as one of the most famous, efficient and the easiest method for implementation.

Principal variation search or **NegaScout** is a directional search algorithm for computing the minimax value of a node in a tree. It dominates alpha-beta pruning, because it will never examine a node that can be pruned by alpha-beta.

NegaScout works best when there is a **good move ordering**. The move ordering is often determined by previous shallower searches. It produces more cutoffs than alpha-beta by assuming that the first explored node is the best.

In **chess engines**, NegaScout has typically given a 10 percent performance increase.

On the Figure 2 illustrated example of the PV working on the game tree.

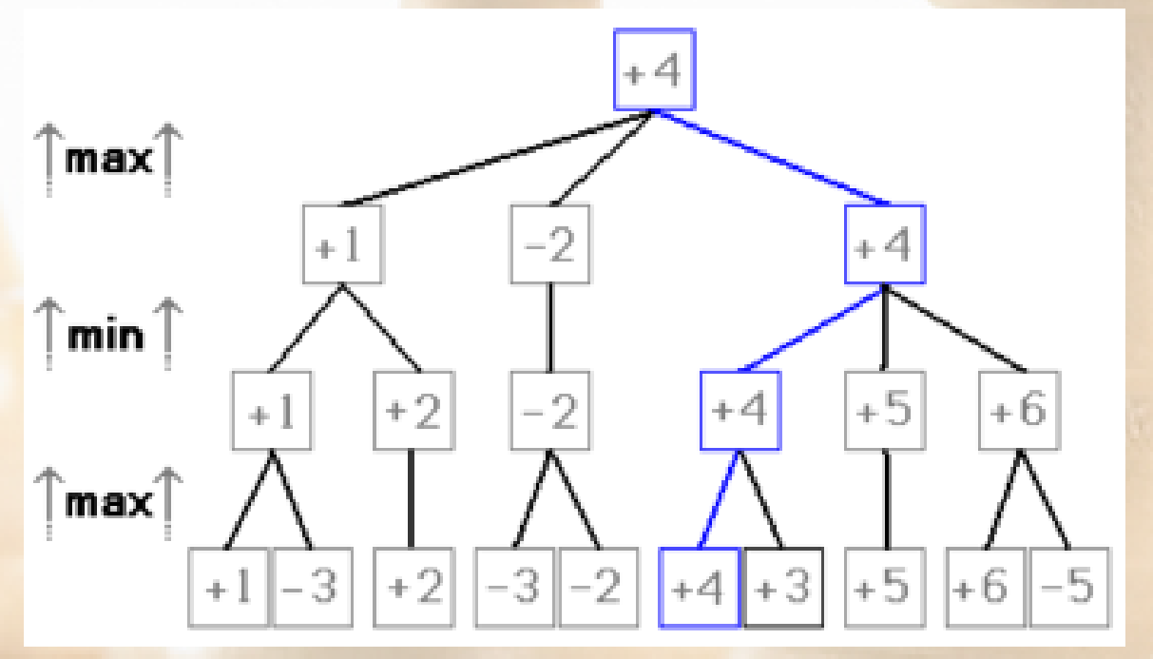


Figure 2: Game tree, PV of a Minimax tree in blue.

Comparison with other methods

To compare search time with other methods (alpha-beta pruning and brute-force search) we took several random middlegame positions and **measured average time** for searching best move with **depth from 1 to 4**. Obtained results show that PVS is the fastest one.

On the Figure 3 illustrated comparison of methods, mentioned above.

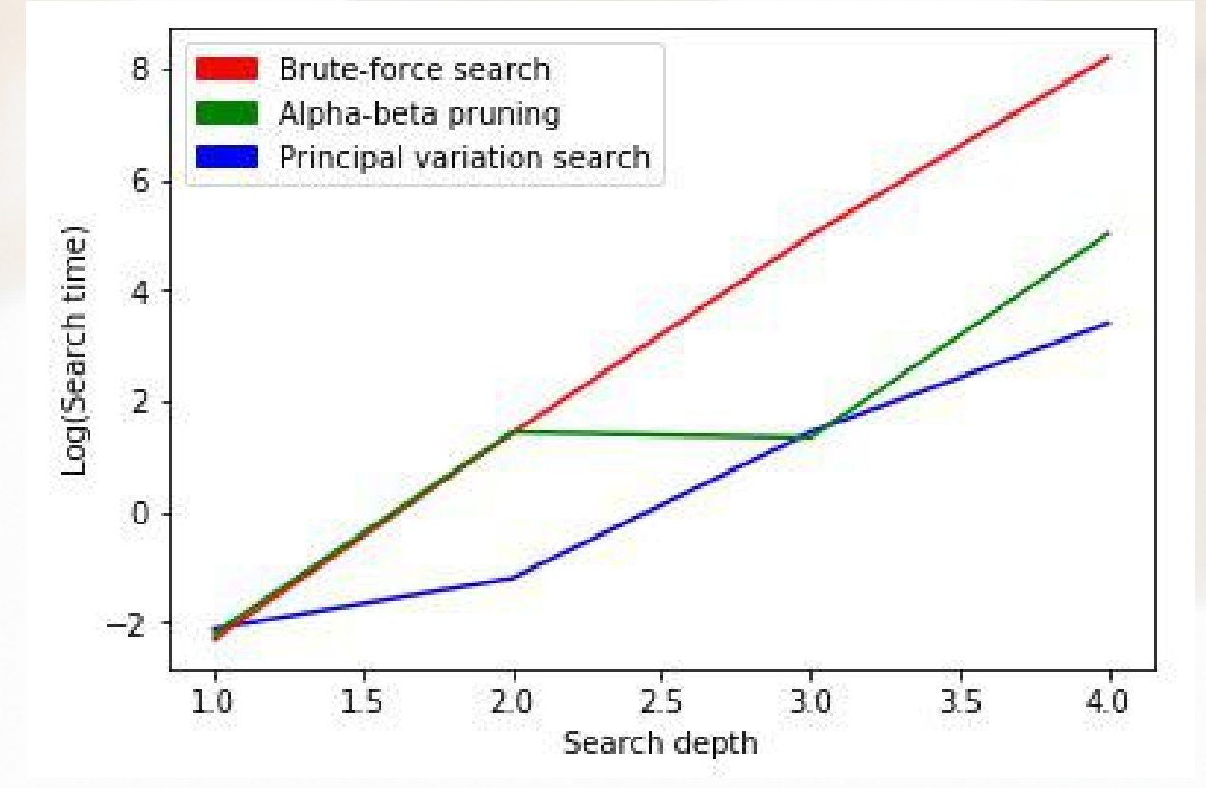


Figure 3: Methods comparison.

Static board evaluation

For performing search among nodes in game tree, there should be a function that estimates score of particular node, which means - to evaluate board state after particular move. Correct estimation of node is crucial for all searching algorithm.

Chess is complex game, there are **a lot of factors** that define which player has better position. The simplest approach takes into account only figures, but not their relative location: pawn costs 100, knight costs 300, bishop - 350, rook - 500, queen - 900, king - infinity.

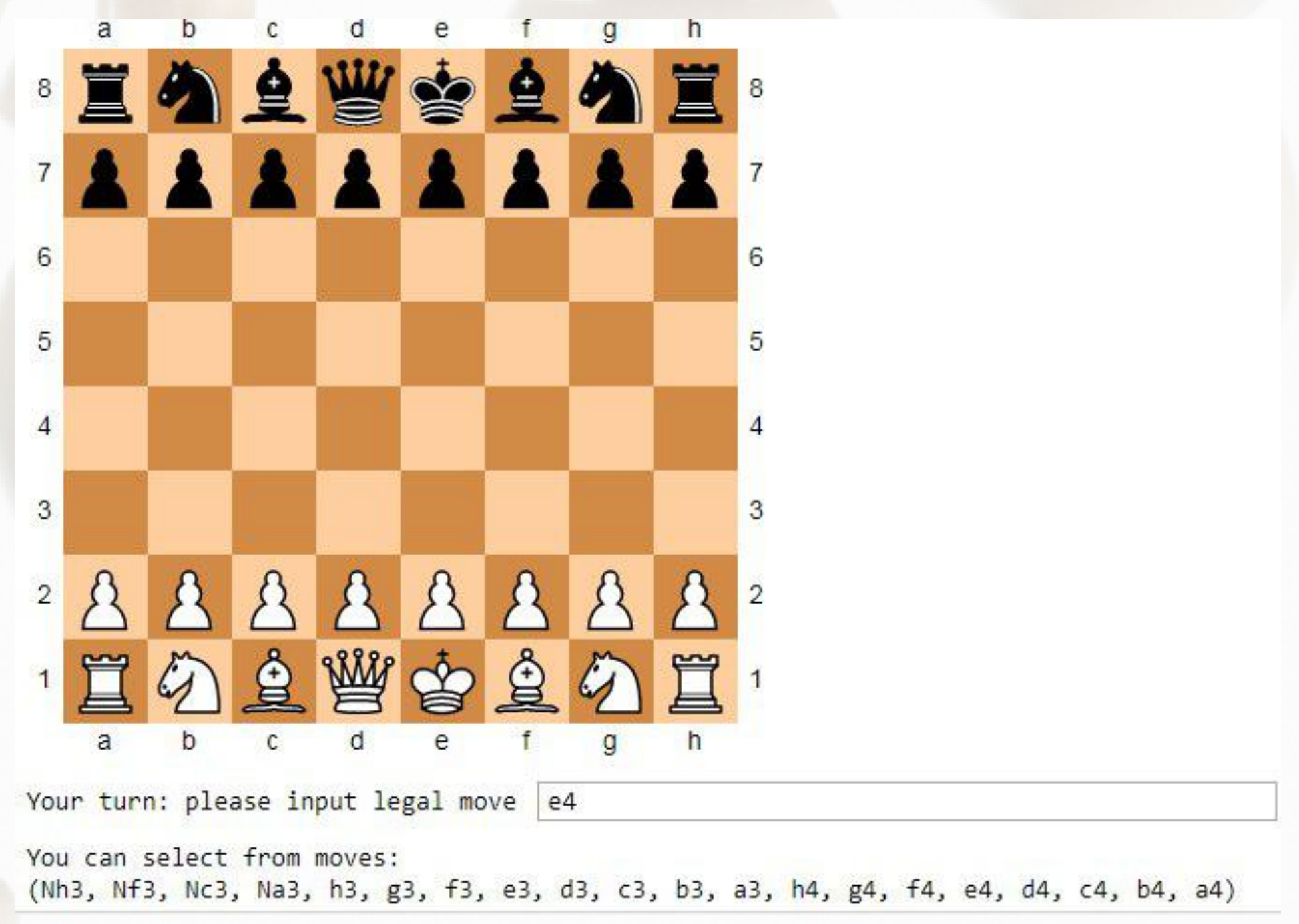
We also added next location factors in function:

1. The closer pawn is to **promotion**, the more is cost.
2. Less amount of players figures on **last horizontal** increase score (especially important in opening).
3. More possible moves you have - higher your score; this factor shows **activeness of figures**.
4. More your figures in centre - higher your score; you have **more space**.
5. More figures you can **attack** and more your figures you can **defend** - higher your score.

All of these factors together with material situation were weighted with some empirical constants. Static board evaluation was calculated as proportion of scores for white and black.

Results

We developed chess game, where user can play against AI. On the Figure 4 presented **started state** of the game board.



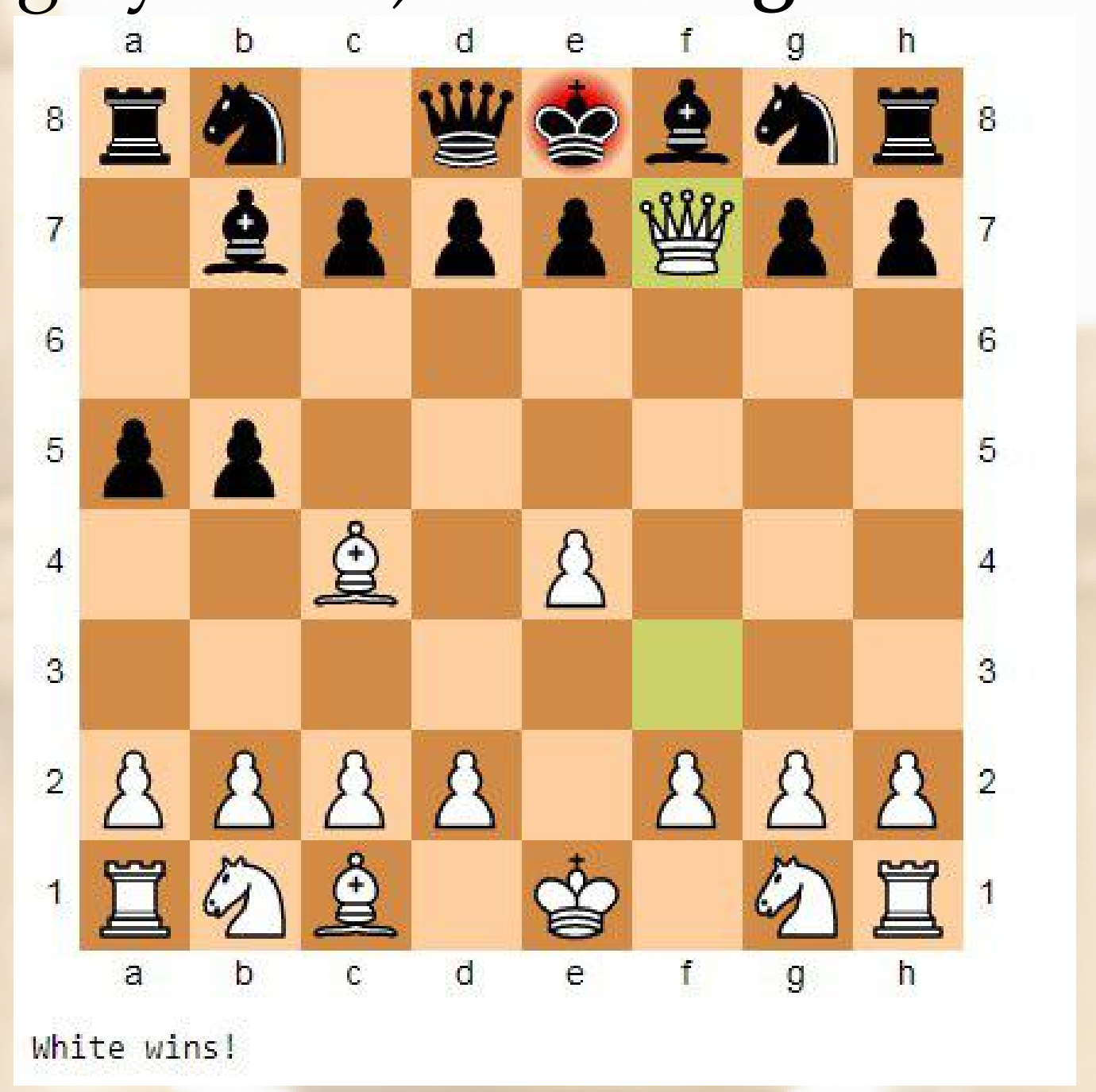
```
Your turn: please input legal move  e4

You can select from moves:
(Nh3, Nf3, Nc3, Na3, h3, g3, f3, e3, d3, c3, b3, a3, h4, g4, f4, e4, d4, c4, b4, a4)
```

Figure 4: Start of the game.

User can input next move in text box in form of special chess notation. Below text box presented all possible moves for user from current position in correct form. After each move program checks conditions of end (mate, stalemate, lack of materials, 75 moves rule, three times repetition), and, if yes, stops game and output message (Draw! White/Black wins!).

On the Figure 5 illustrated final of the game, when user (playing by white) **won the game**.



```
White wins!
```

Figure 5: Start of the game.

Discussion

The main and most important problem for developing successful AI in chess is proper static board evaluation.

Best chess engines also use **magic constants** to balance impact of different factors, but they are more properly tuned than in this program (where constants were chosen only based on authors experience). Obviously, known engines take into account larger number of factors.

What is more, they also use chess **database** for better moves in opening and specially developed **algorithms for endgame**.

References

1. chessprogramming.wikispaces.com/Principal+variation
2. en.wikipedia.org/wiki/Principal_variation_search
3. chessprogramming.wikispaces.com/Chess
4. pypi.python.org/pypi/python-chess
5. www.gamedev.net/articles/programming/artificial-intelligence/chess-programming-part-v-advanced-search-r1197

Contact

Github repository of our project:
https://github.com/dil-delada/algo$_{project}$

Advisors: Jaak Vilo , Dmytro Fishman