LPP course

Chess program

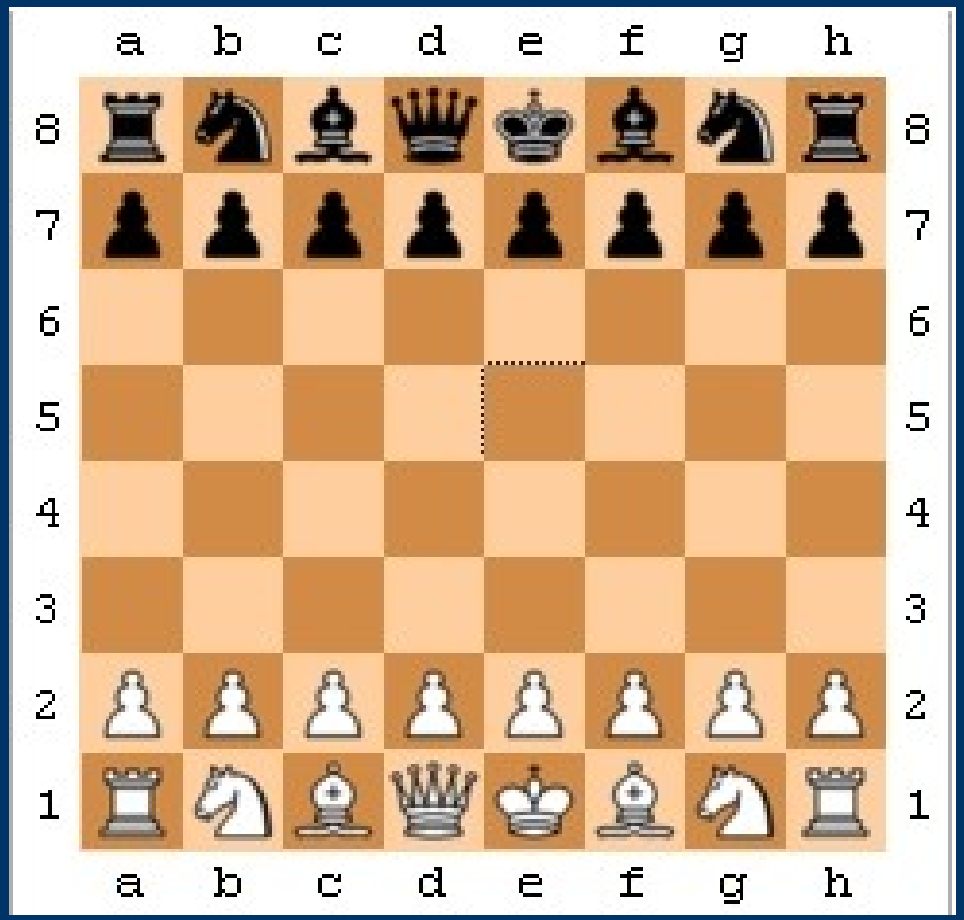
Katarzyna Obrycka, Marek Chruściel

Agenda

- Chess moves rules.
- Our approach to problem.
- Project design and work.
- Conclusions.

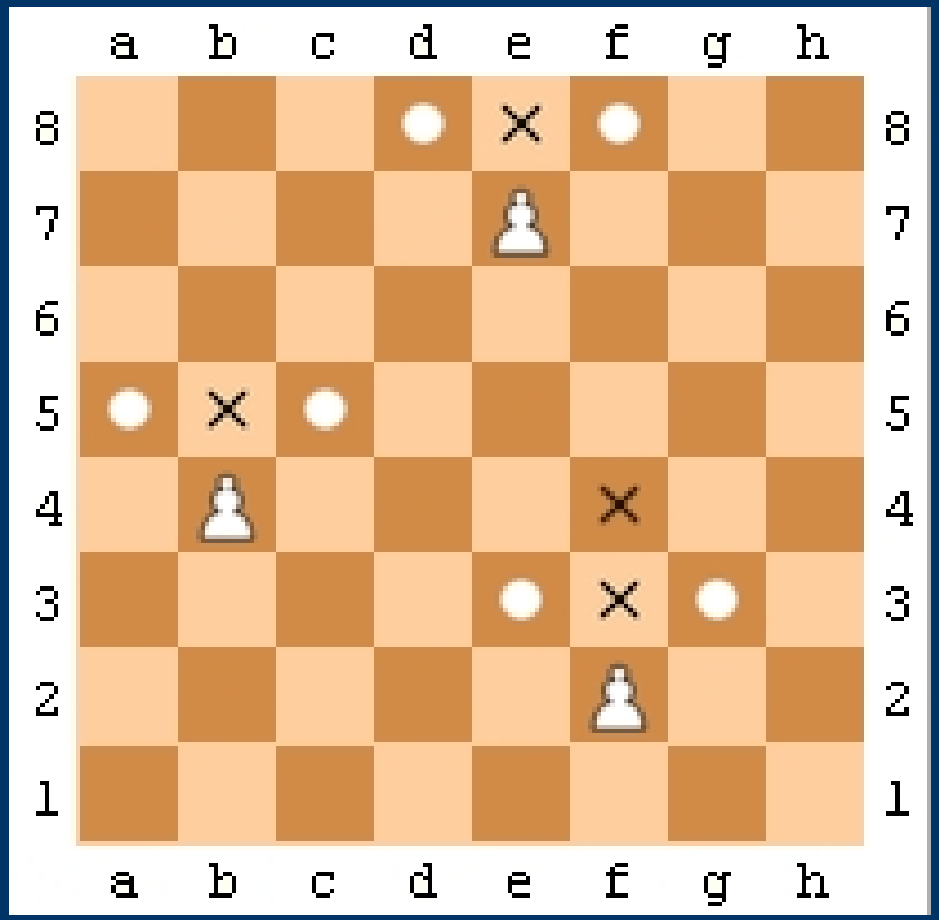
Chess moves rules

The initial board



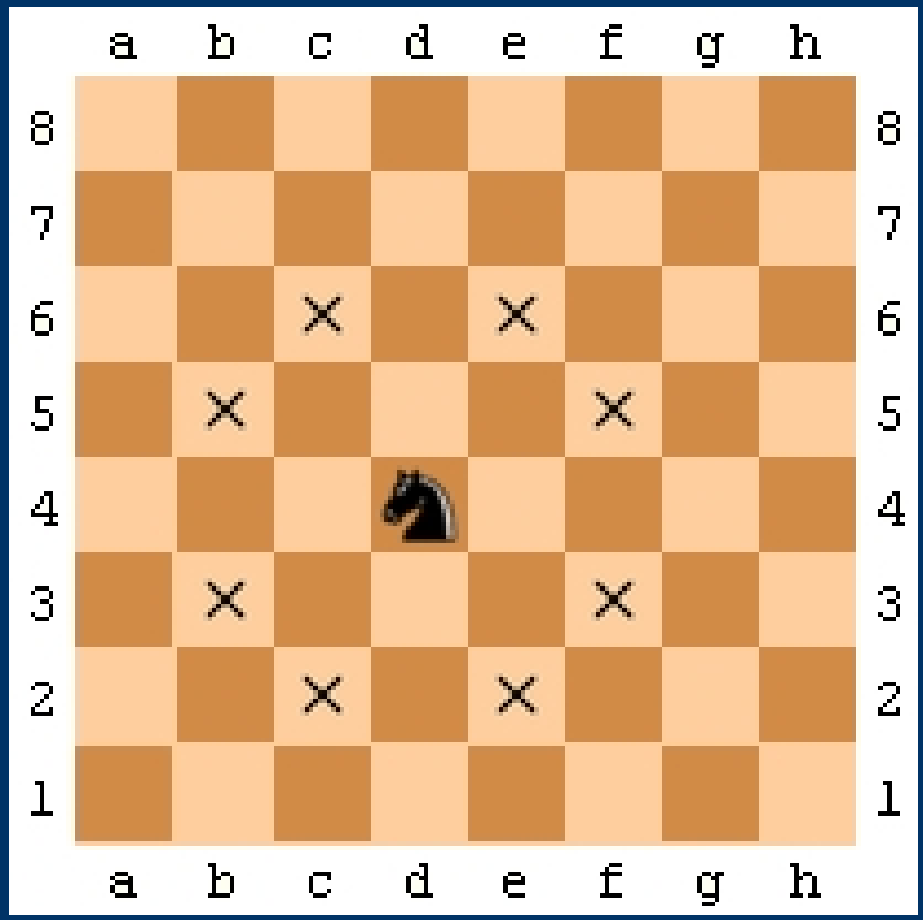
Chess moves rules

Pawn moves



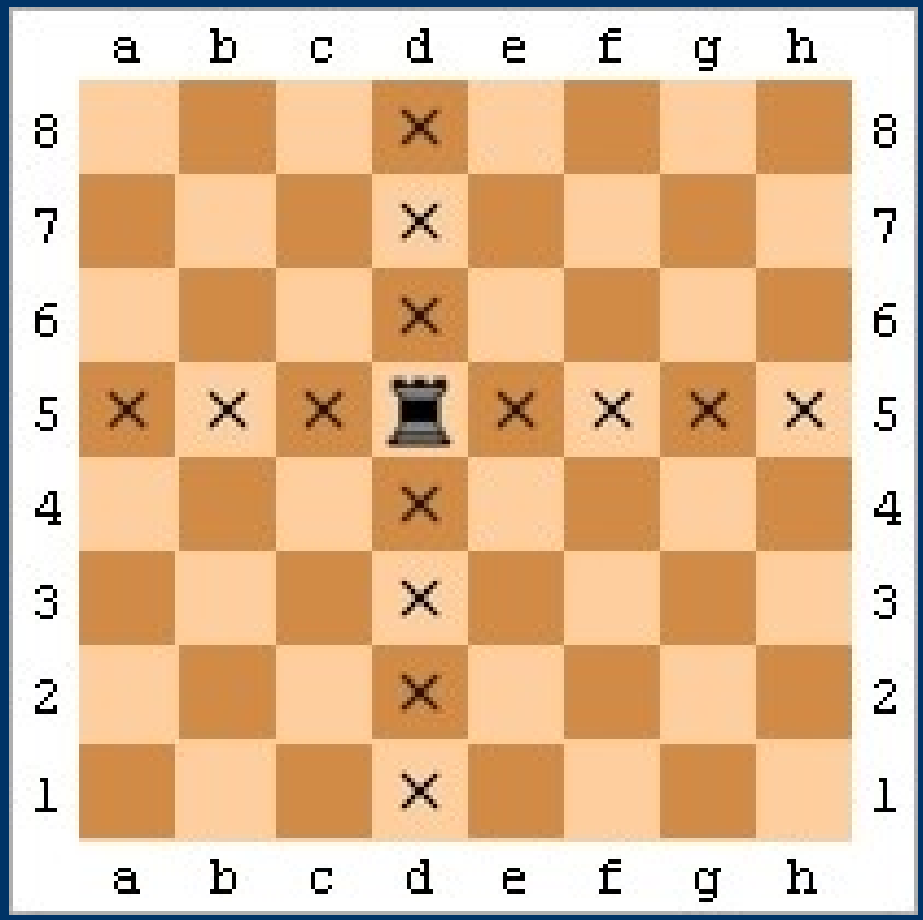
Chess moves rules

Knight moves



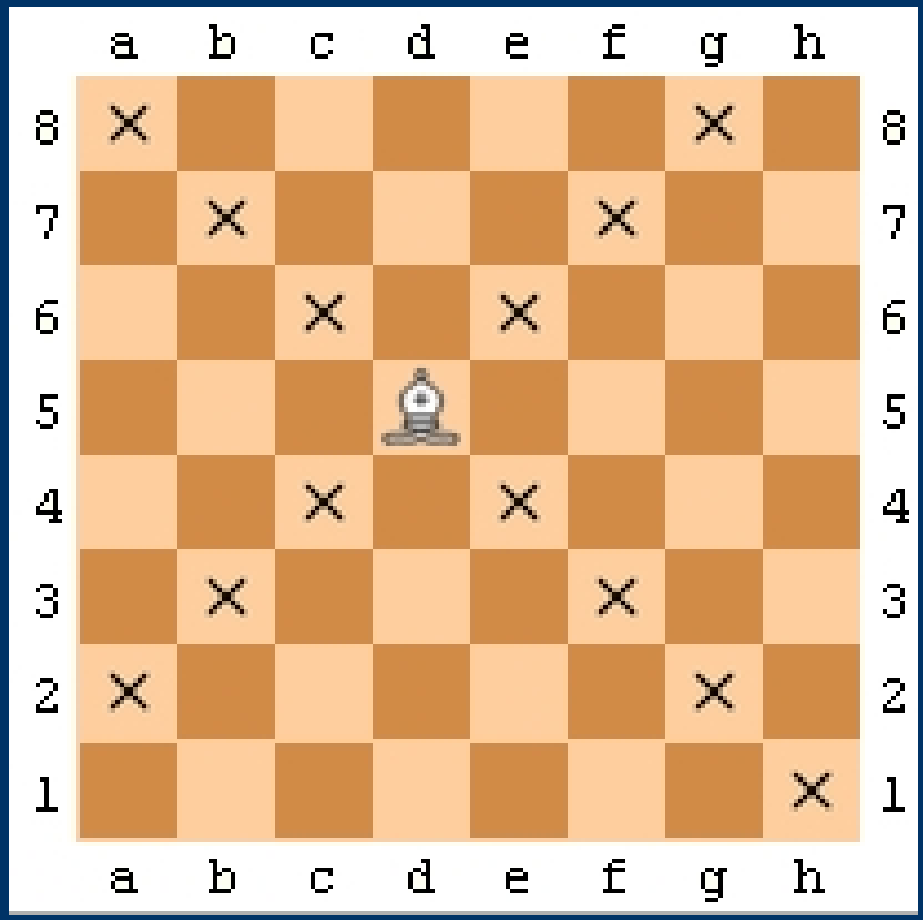
Chess moves rules

Rook moves



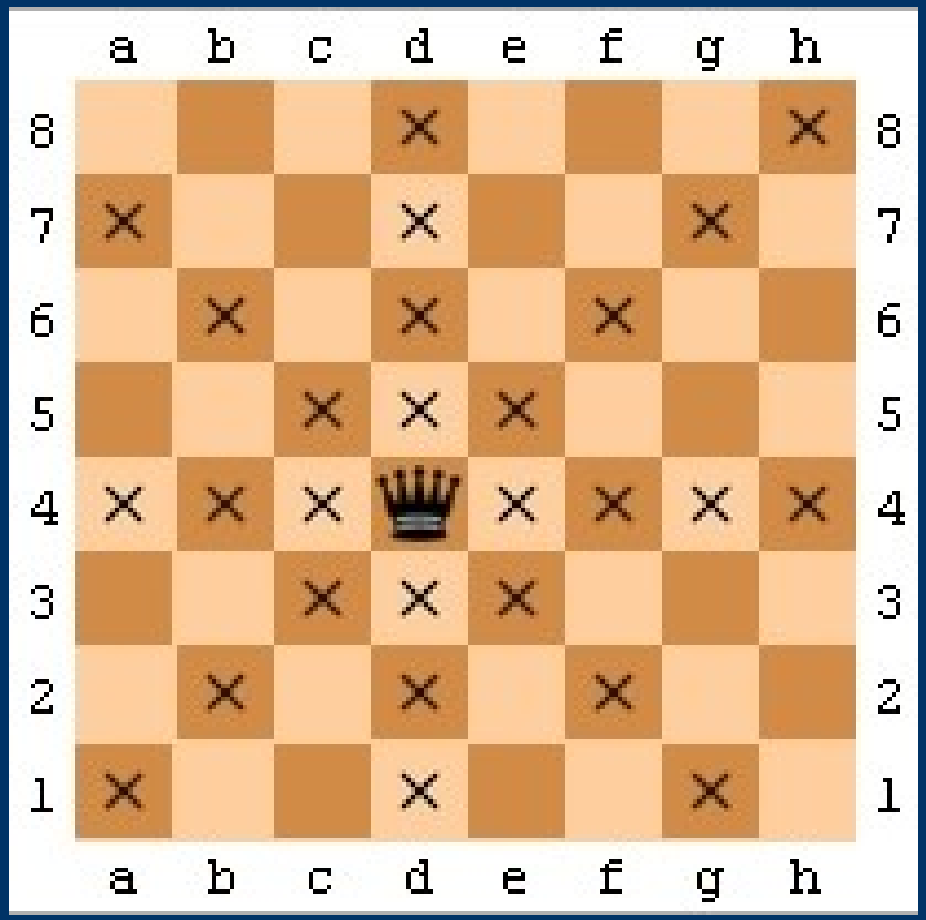
Chess moves rules

Bishop moves



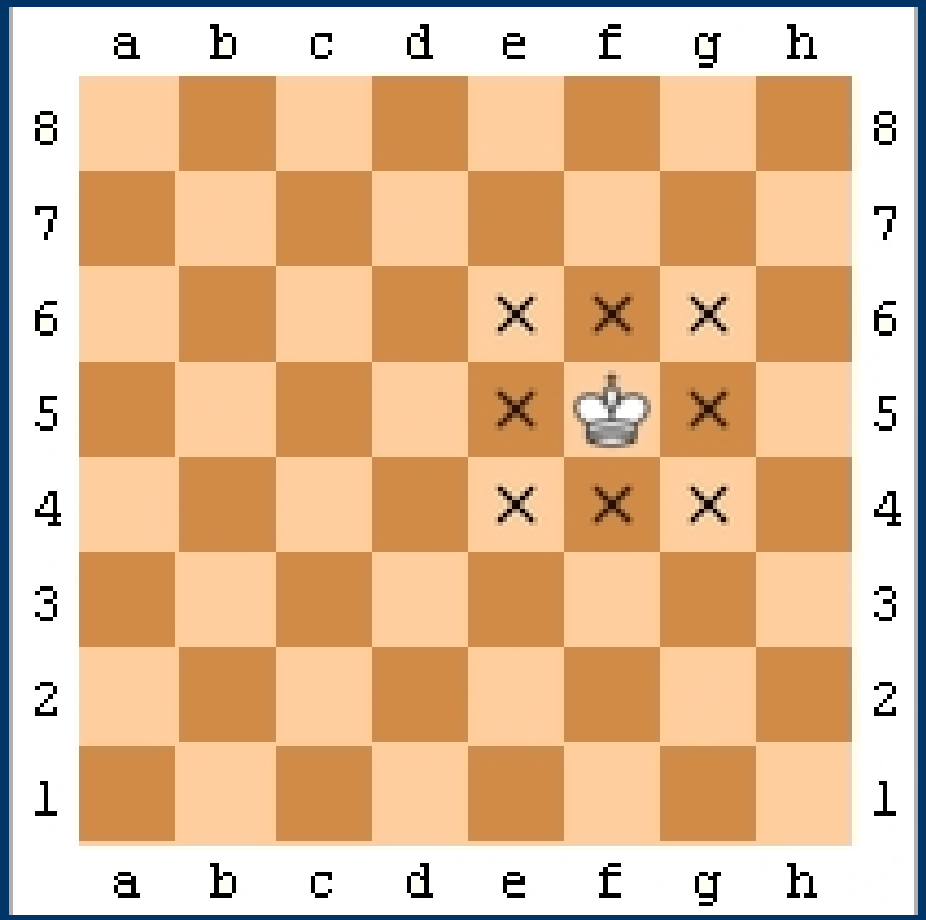
Chess moves rules

Queen moves



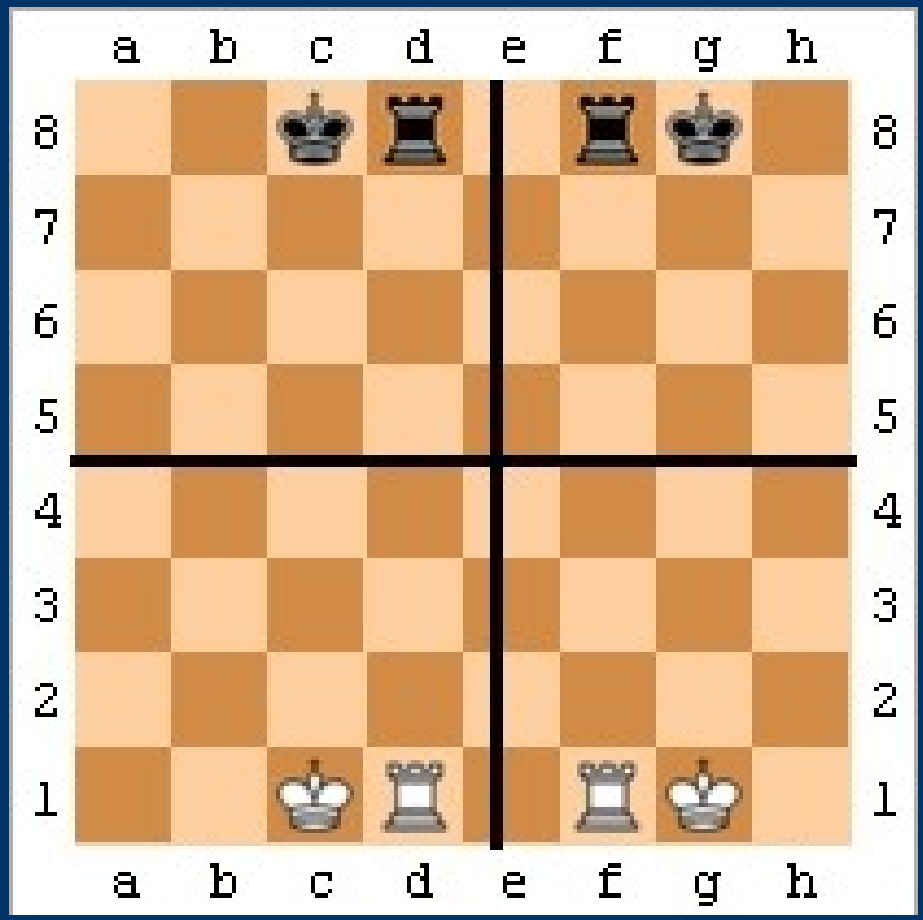
Chess moves rules

King moves



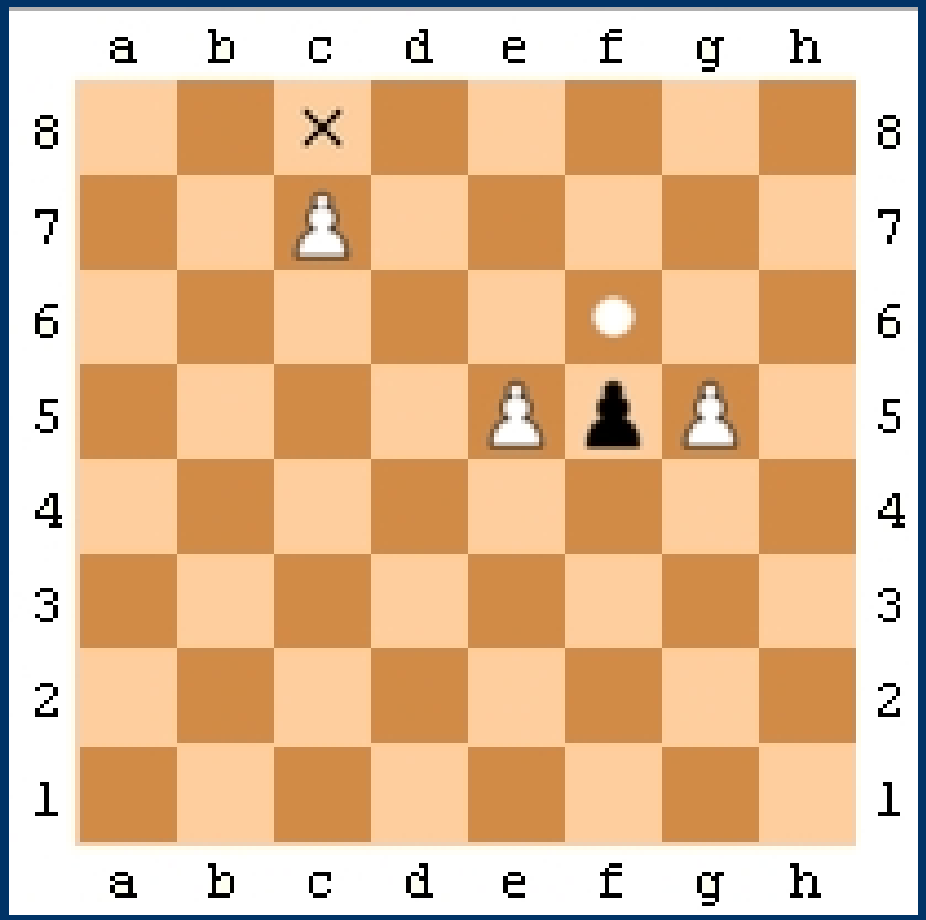
Chess moves rules

Castling – special move



Chess moves rules

En passant, promotion



Our approach to problem

- Minimax algorithm for getting the best possible move. Standard algorithm, every chess program uses it in some way.
- “DAG” data structure that stores the game tree. Main reason why to remember the game tree is to compute and evaluate only once.
- Game tree holds game states in nodes and moves in edges.

Our approach to problem

- Evaluation function that uses piece counting as main evaluation method. Also pieces mobility and checks are considered.
- Piece counting is the main method for evaluating the game state in every chess evaluation function.
- Our goal was to learn something about artificial intelligence in games.

Project design and work

- “DAG” - class that stores the game tree and has the implementation of minimax algorithm.
- GameState – class that stores information about one particular game state.
  It's functionality: apply a move, compute checks, and provide all information about the state.
- Move – structure that hold information about one move.
- Move generation – generates all possible moves from one state.

Project design and work

- EvaluationFunction – class that evaluates one GameState. As mentioned it uses the piece counting as main method.
- Xboard – provides linking form Xboard engine to our program.
- Util – helper functions.
- Test – some tests, used as a “sandbox”

Conclusions

- We made a chess program that plays chess in some decent way.
- The implementation is much harder than it sounds.
  - The program must work fast
  - Move generation is complex
  - Game openings
  - Pondering
  - Check and mat computing
- This leads into a lot of code.
- Our architecture was not the best one.

Conclusions

- Students always have no time for making projects.
- Finding bugs is (nothing new) annoying.
- C++ is not the best choice until you know it well.