The Cross-Platform Implementation Of Draughts Game

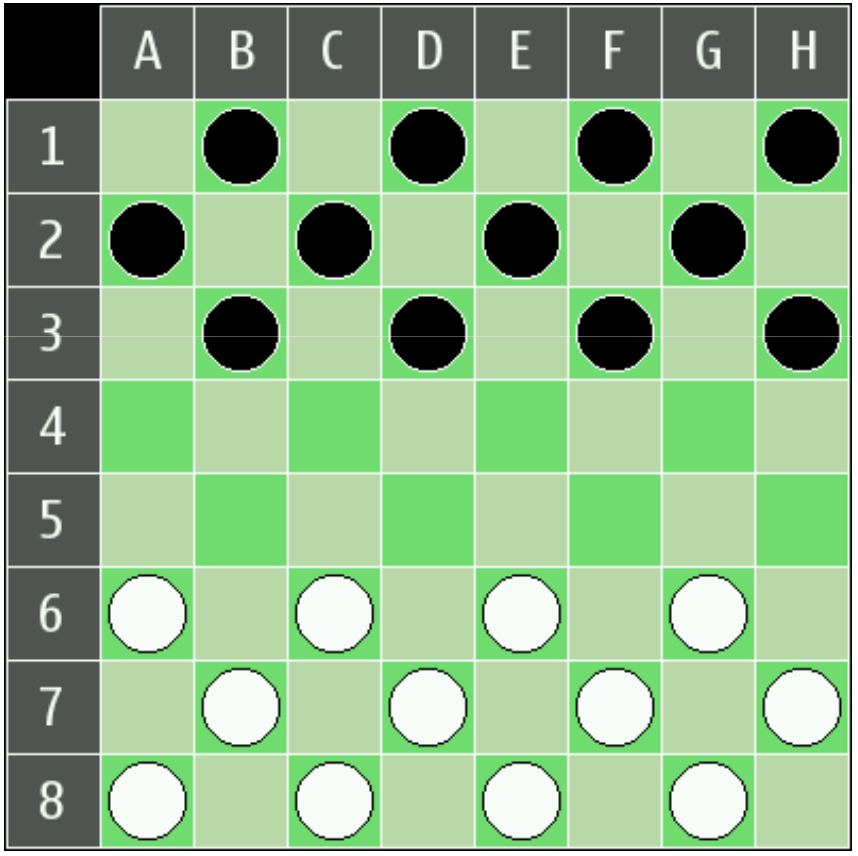
Valery Kirkizh

State University of Aerospace Instrumentation

Saint-Petersburg, Russia

Introduction

- Age-old Russian game Draughts;
- Cross-platform implementation, Qt;
- Artificial intelligence algorithms;



The program functionality

- Full game process;
- Artificial intelligence;
- GUI written on Qt4;
- Cross-platform working;

Implementation details

- C++ (productivity), Qt4 (cross-platform possibilities);
- MVC pattern:
  - Model: the Board, the Game;
  - Several Views;
  - Two Players;

Model

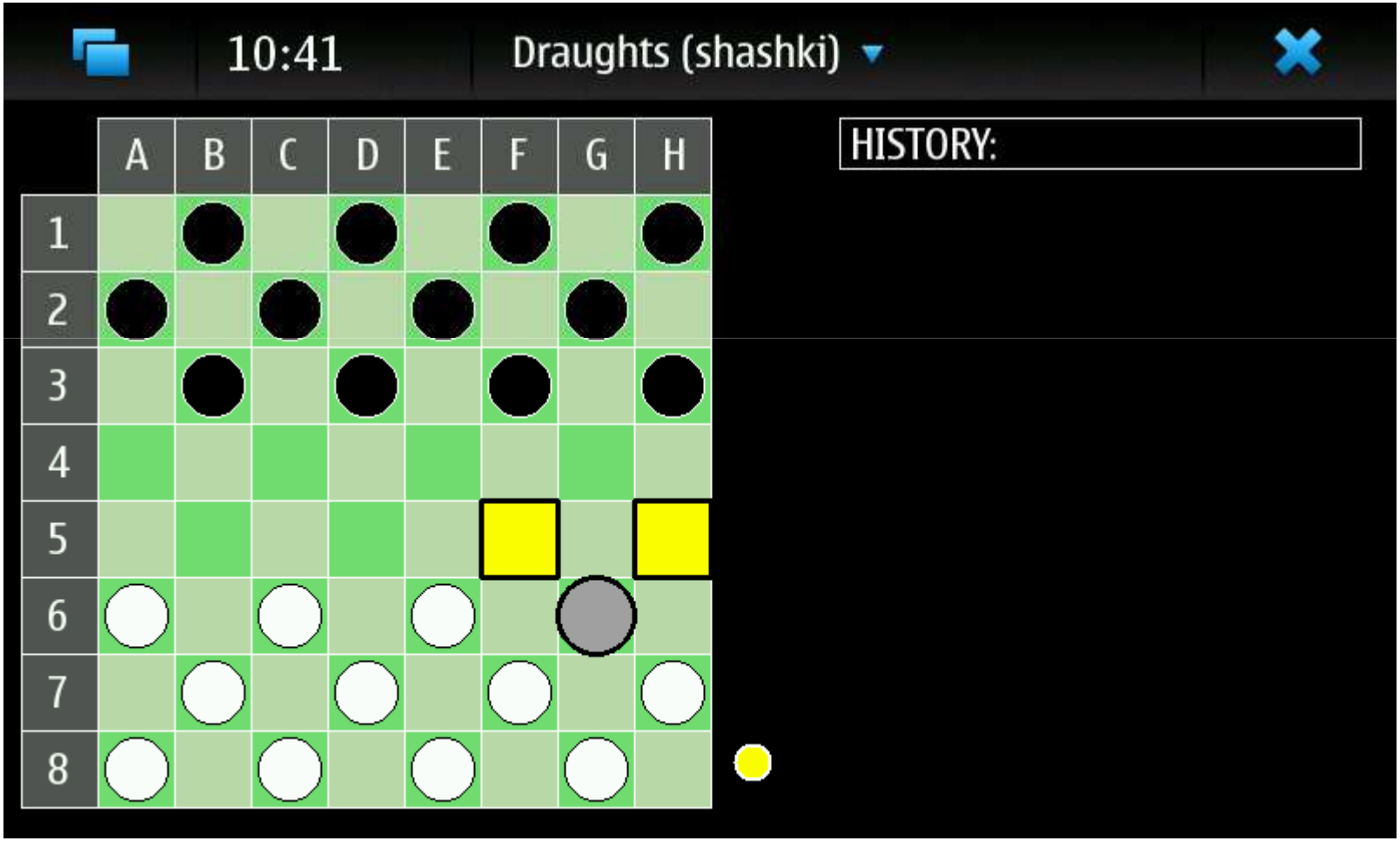
- The Board:
  - Do key game actions;
  - Checking moves possibility;
  - A draw tracking;
- The Game:
  - Control the game position;
  - Watches the game situation;
  - Contains the Board and the two Players;

Views

- Several Views:
  - Start game View;
  - Board View;
  - Information View;
  - Finish game View;
- Observer pattern
  - View is observer of the Game;
  - Views are independent;

Start game interface

Board and information



Players

- Two kinds of players:
  - Human player;
  - Computer player (AI);
- Human player is a Board View;
- Computer player is based on abstract factory pattern
  - different AI algorithms;

Platform differences

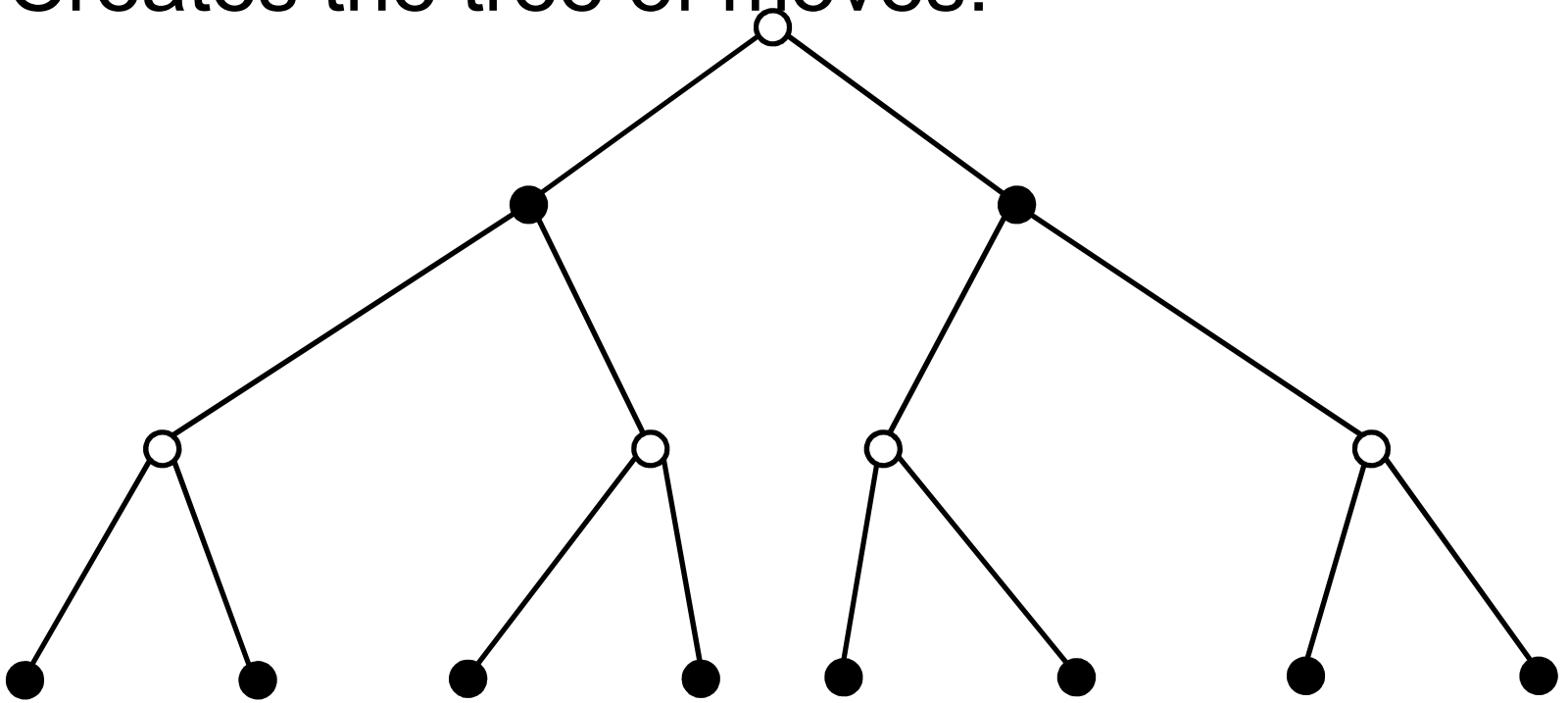
- Two main stumbling blocks
- Graphic user interface
  - QT, not platform-dependent stuff;
  - “#ifdef” instructions in confusing places;
- Multi-threading possibility
  - is not supported in mobile linuxes;
  - works only in PC version;

Artificial intelligence

- Algorithms:
  - NegaMax;
  - Alpha-beta prunning;
  - NegaScout;
- Exhaustive search and its improving;

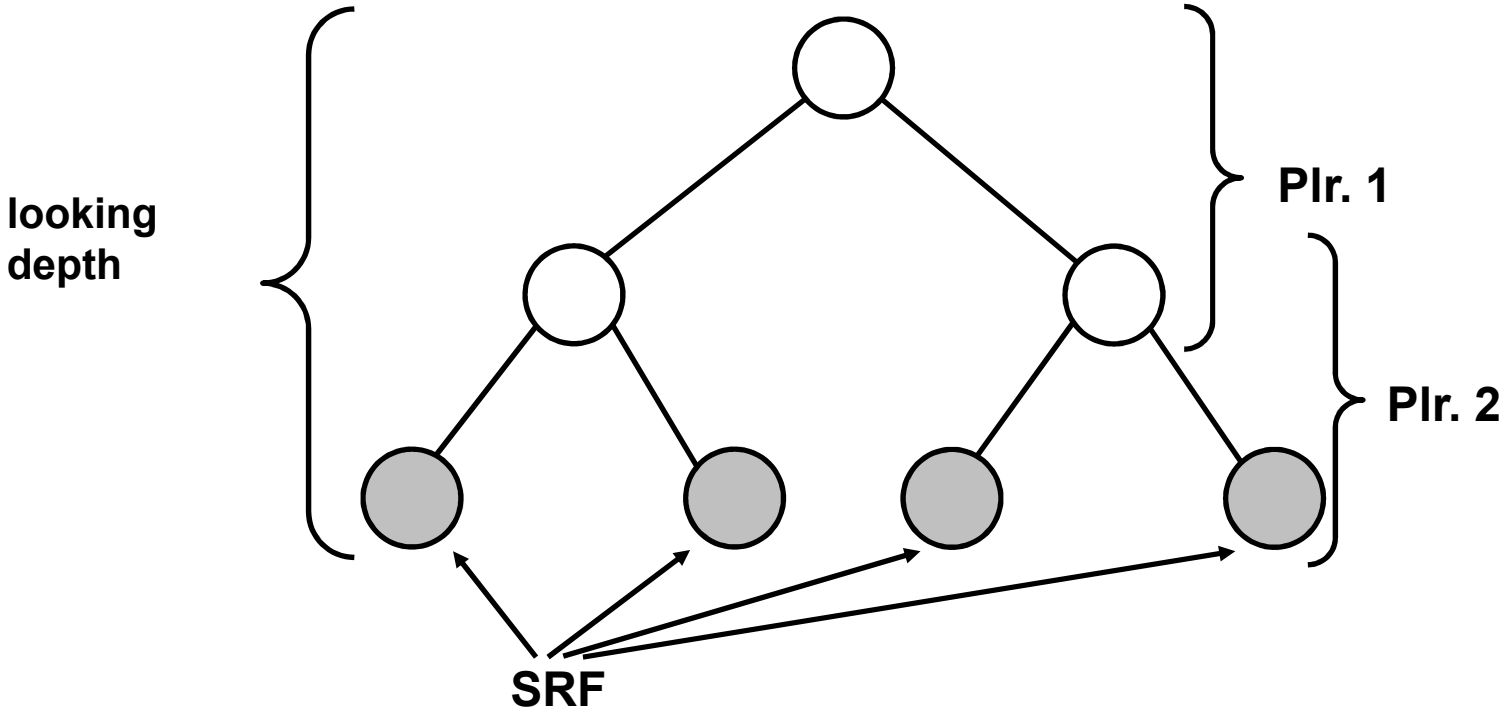
NegaMax algorithm (1)

- Creates the tree of moves:



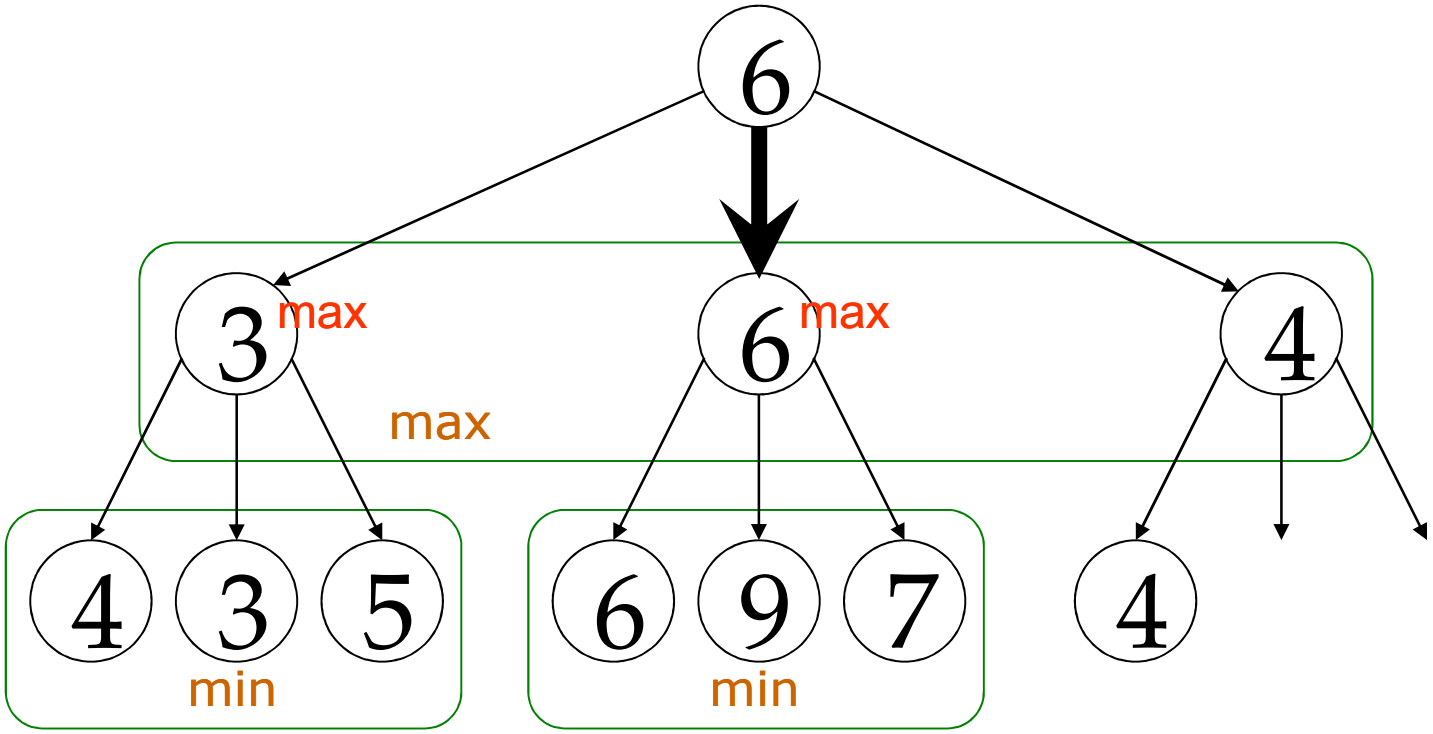
NegaMax algorithm (2)

- Chooses the best move:



Alpha-beta prunning

- Uses branch and bound method for cutting off unnecessary nodes of the tree:



Time control

- There is time control in real games;
- Each player gets a certain amount of time;
- Players are free to decide how to spend their time;
- Need to develop time control strategy for players;

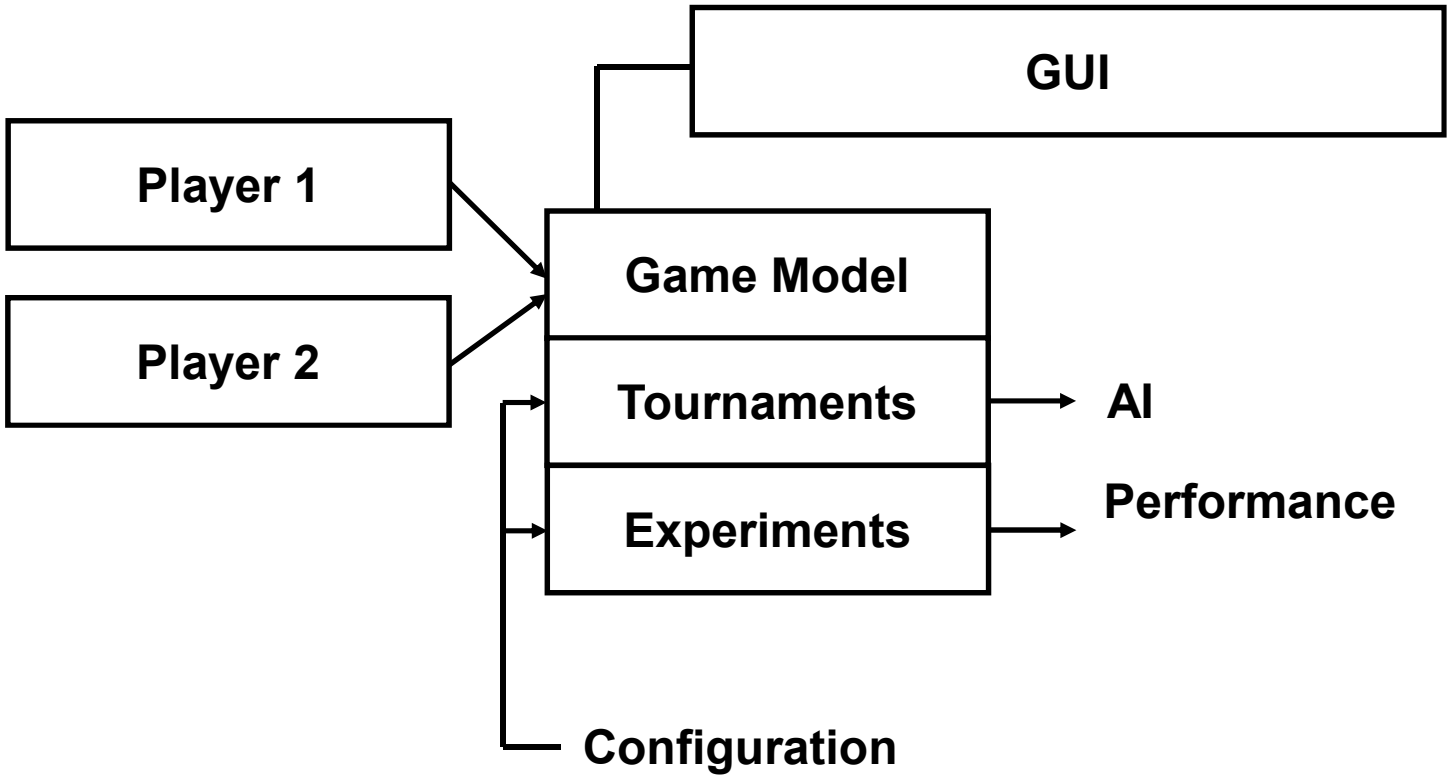
We set following tasks:

- choose the fastest algorithm:
  - NegaMax;
  - AlphaBeta;
  - NegaScout;
- choose the best time control strategy:
  - Simple;
  - Defensive;
  - Aggressive;

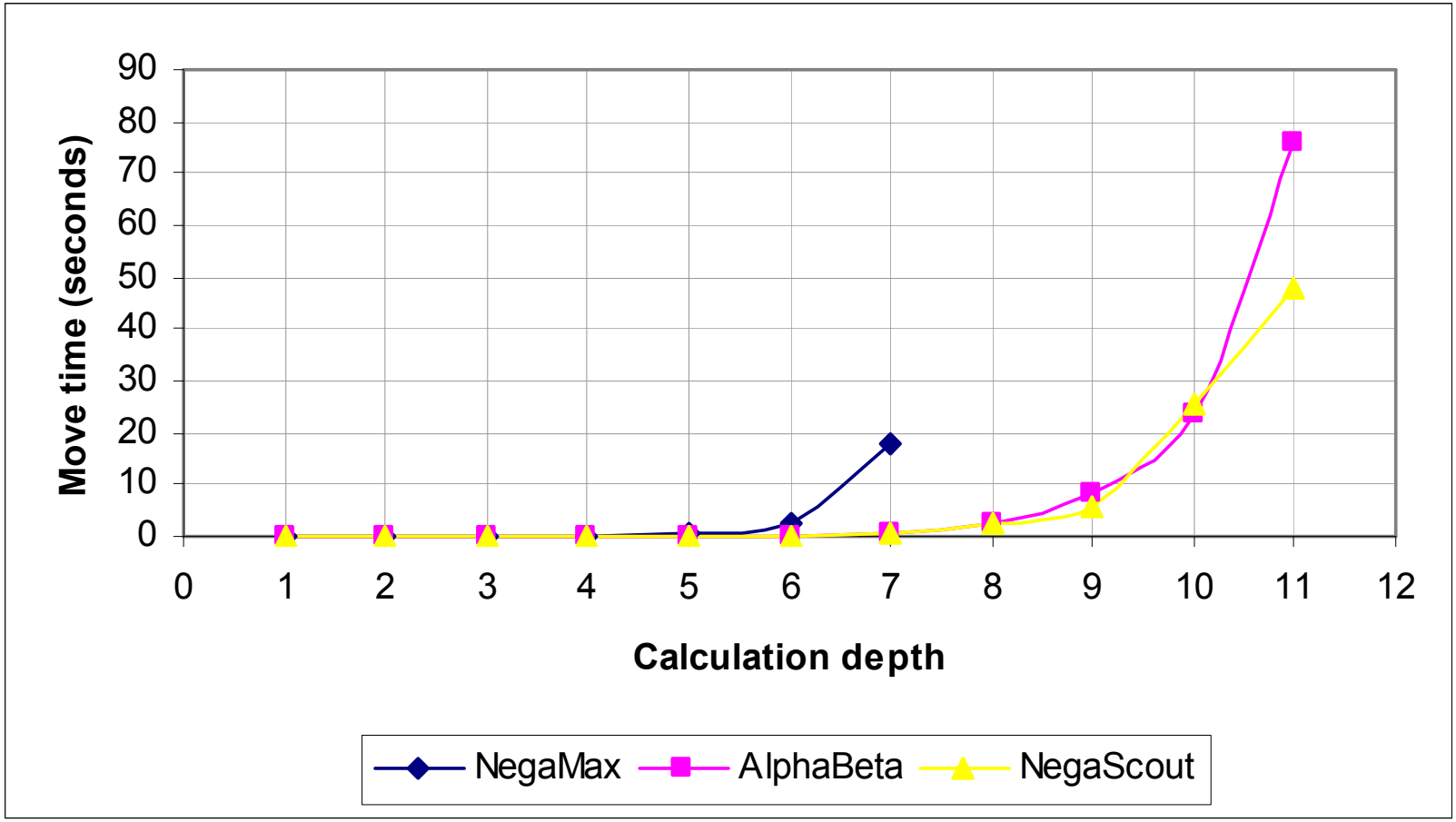
Test bench

- Special program complex;
- Performance comparison – “Experiments” tool (if time is not limited)
  - NegaScout has the best performance, NegaMax – the worth;
- Intelligence comparison – “Tournaments” tool (if time is limited)
  - NegaScout has the best intelligence (it can try a larger number of moves);

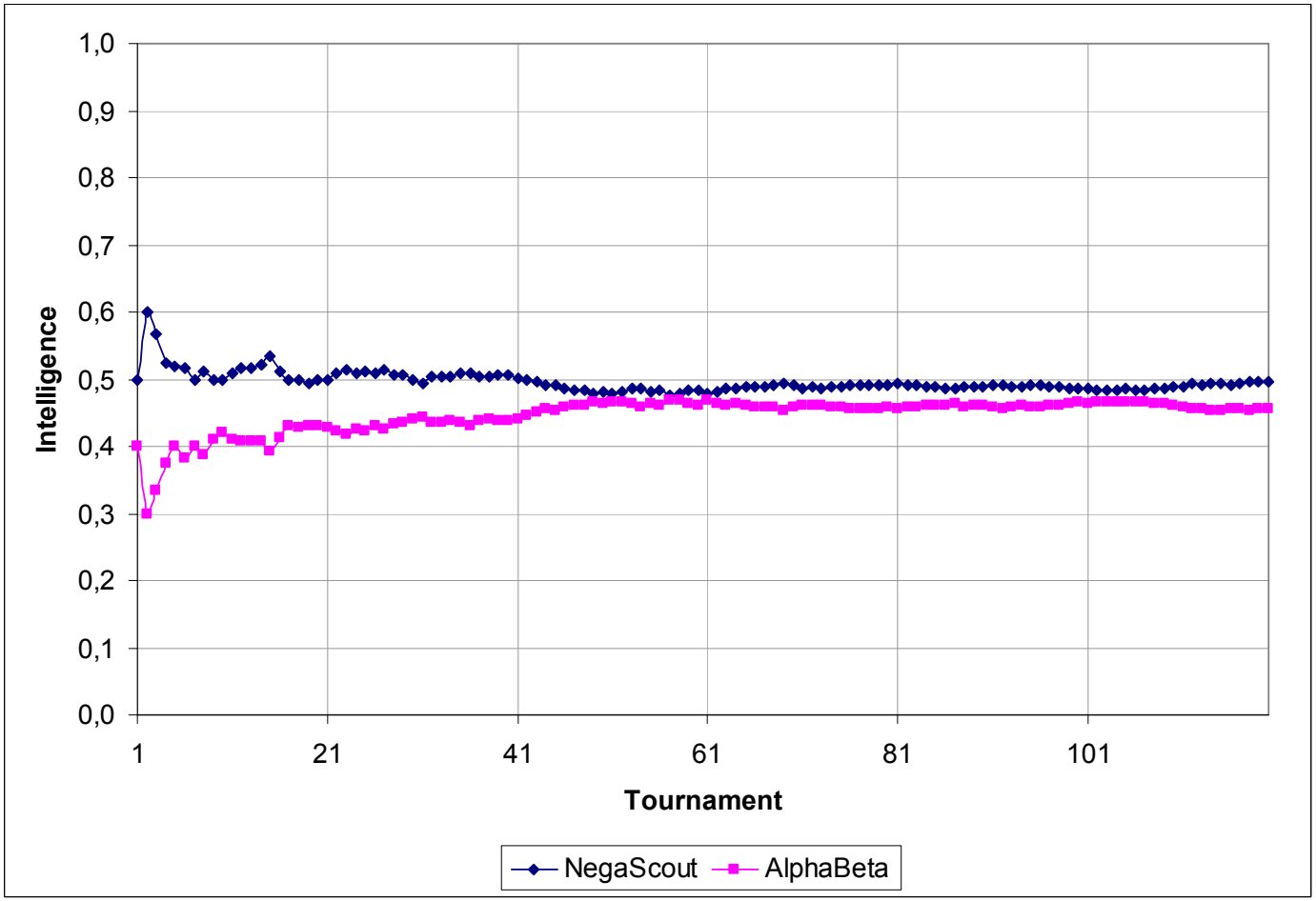
Program complex scheme



Testing algorithms' performance



Testing algorithms' intelligence



Time control strategies

- The simple strategy
  - uses strictly specific amount of time on the move;
- The defensive strategy
  - saves time at the end of the game;
- The aggressive strategy
  - leaves at the end of the game more time;
- The best strategy is Defensive
  - because the comparisons were in quick tournaments due to lack of time;

Conclusion

- Results:
  - Fully functional cross-platform game;
  - Testing program complex;
    - NegaScout has the best performance and the best intelligence among considered algorithms;
  - Source: http://code.google.com/p/shashki/
- Future directions:
  - AI improvment, neural network addition;
  - Time control strategies extension;