The Cross-Platform Implementation of Game Draughts

Valery Kirkizh, Evgeny Linsky
State University of Aerospace Instrumentation
St. Petersburg, Russia
{vkirkizh, evlinsky}@vu.spb.ru

Abstract

Draughts is an age-old Russian board game for two players. In this paper we describe the cross-platform (based on Qt) implementation of this game, which works on mobile Linuxes. Our program supports as playing of the two human players, as playing with a computer. The paper describes the computer player and the details of implementation.

Index Terms: Artificial intelligence, games, Qt, Maemo.

I. Introduction

Nowadays almost everybody knows about an age-old Russian game Draughts. It is a board step-by-step game for two players. A basis of this game is a board with some figures – whites and blacks (see fig. 1).

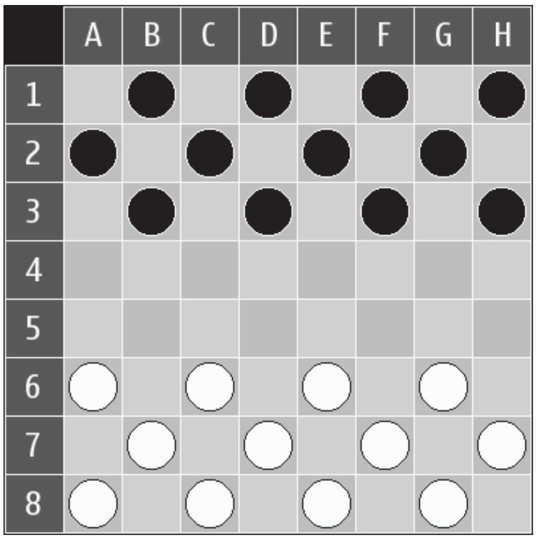

Fig. 1. The board of the game Draughts: starting position.

A white player moves first. Then the game continues step by step until one of the players loses possibility to move his figures.

Our task was to develop cross-platform program with possibility to play with human or computer player, working under desktop Linux (e.g. Ubuntu) and mobile Linuxes (Maemo, MeeGo) using Qt.

The functionality of the game includes following features:

- Full game process (starting, playing, finishing, time control) with game rules control and tips for players;
- Artificial intelligence for playing with computer;
- GUI written on Qt4;
- Cross-platform working.

The paper has the following structure. The section II provides details of implementation. The section III describes a computer player.

II. Implementation details

An artificial intelligence (AI) in such games is based on exhaustive search, which usually takes a lot of time, so C++ was chosen as language of the implementation. For creating of cross-platform GUI we chosen Qt library because it is supported on many platforms including mobile Linuxes (Maemo, Meego).

The architecture of the program is based on MVC pattern. The Model contains the rules of the game. It is divided into two parts: the Board and the Game. They are two C++ classes. The Board enables to do key games actions. It has such methods as checking move possibility, move execution, checking if game is continuing or finished, etc. There is an important goal of the Board: a draw tracking. The Board watches white and black figures number. If the number of figures does not change within 32 moves, the Board declares a draw and the game ends. The Board is completely covered by unit tests. The Game watches the game situation. It contains the Board and two players. In the beginning of the move the Game transfers control to current player. The player makes his move after that control is transferred to the Game. The Game checks the correctness of received move and, if the move is correct, executes it. At the end, it checks game state (win, draw, etc). If the game continues this process is repeated.

The GUI on mobile Linux and on desktop PC has some differences. These differences are implemented using “#ifdef” instructions in the source code. Also we would not use platform-dependent stuff. There are several Views of the model including start game view (see fig. 2), board view (see fig. 3) and information view. This scheme is based on the observer pattern. Each view is observer for the Game. As soon as the board is changed, the game notifies its observers about the changing. It enables to develop many independent views.

There are two kinds of the Players, which could change the Model: human player and computer player (AI). A computer player may be based on one of the different AI algorithms. Players’ implementation is based on abstract factory pattern. The Controller links the Model with Players and Views.

One of the differences between desktop PC and mobile devices is multi-threading possibility. It is known that modern computers have two and more cores. That’s why we implemented multi-threading in our program. Firstly, it enables the GUI to operate independently from the calculations (e.g. calculating the next move). Secondly, it allows to perform computations faster. Unfortunately, mobile Linuxes are not supported execution of more than one thread. So this feature is worked only in PC version.

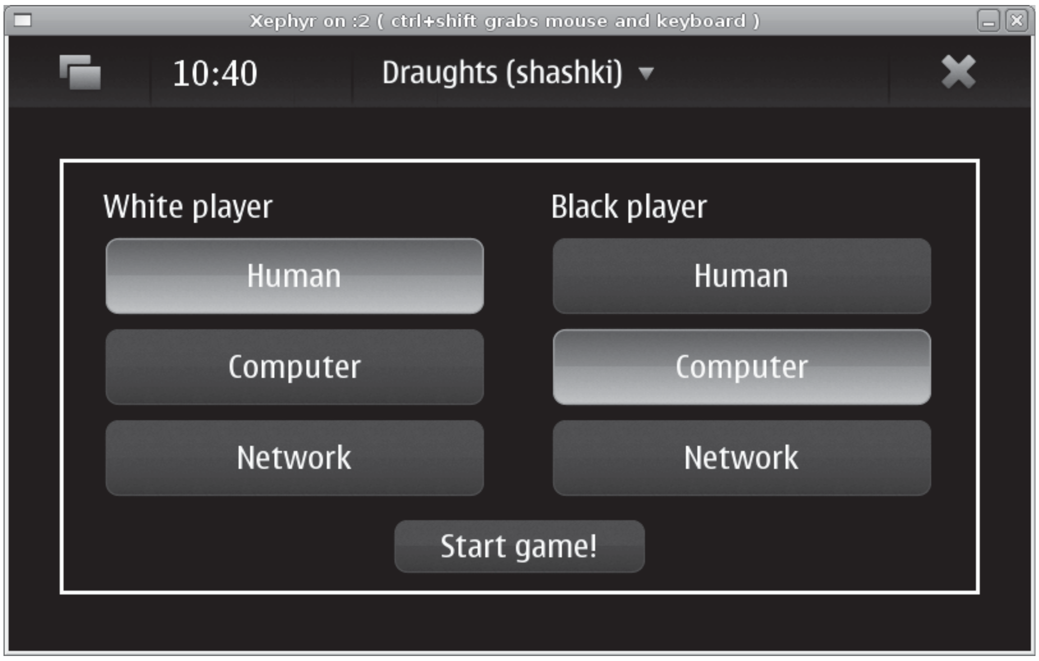

Fig. 2. Start game view

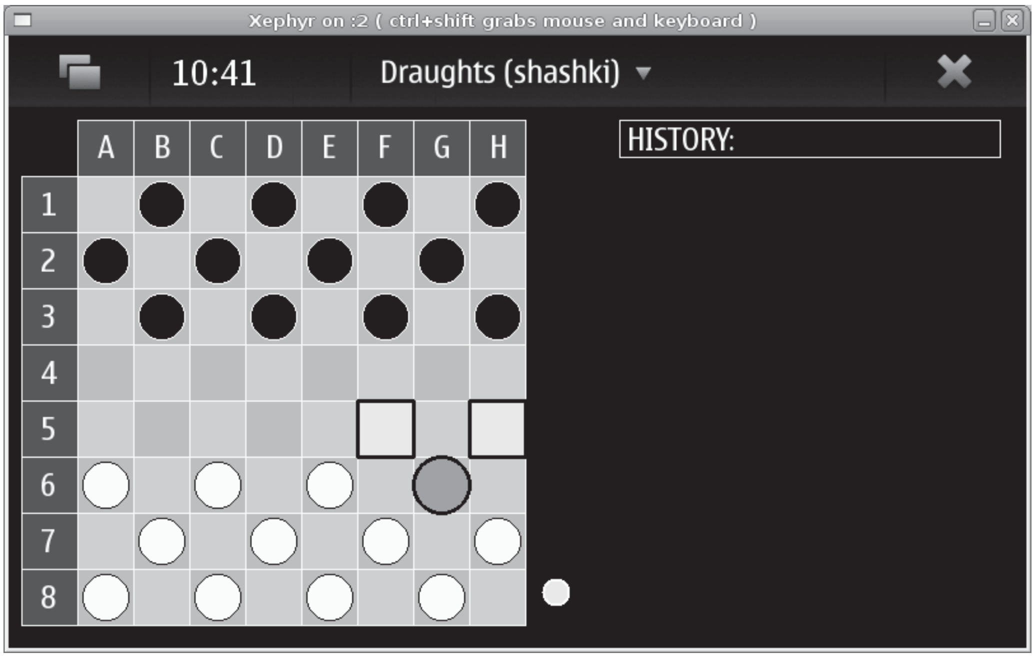

Fig. 3. Board view

III. Artificial intelligence algorithms

In this work we consider the following AI algorithms: NegaMax (full search tree), Alpha-beta pruning (NegaMax improving by branch and bound method) and NegaScout (Alpha-beta improving by search with a zero window).

NegaMax is based on constructing a tree of moves (see fig. 5).

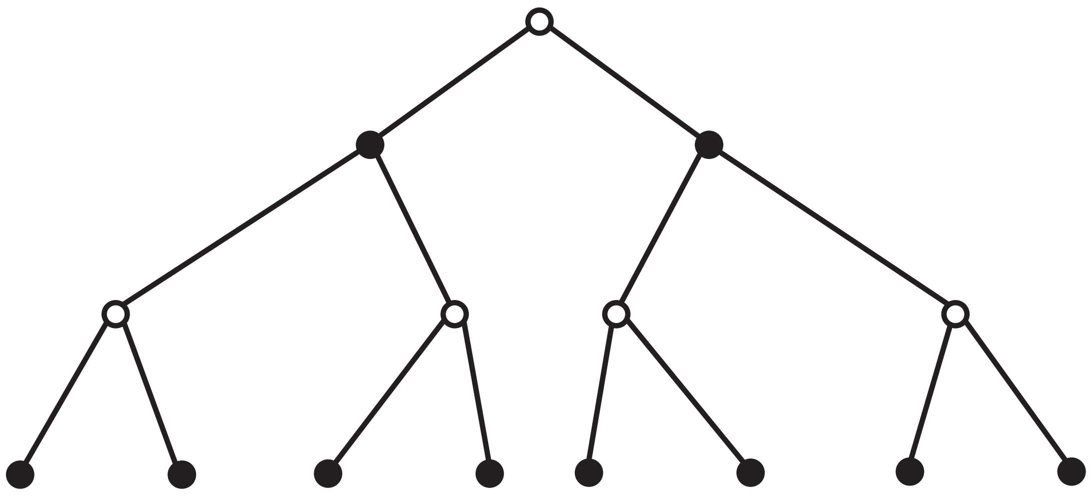

Fig. 5. A tree of moves

After tree constructing this algorithm looks at each node of the tree and finds a game situation assessment. Special statistic rate function (SRF) is called in the terminal nodes of the tree. Then the scores in terminal nodes are converted by calculating maximum of the scores in the node branches (see fig. 6).

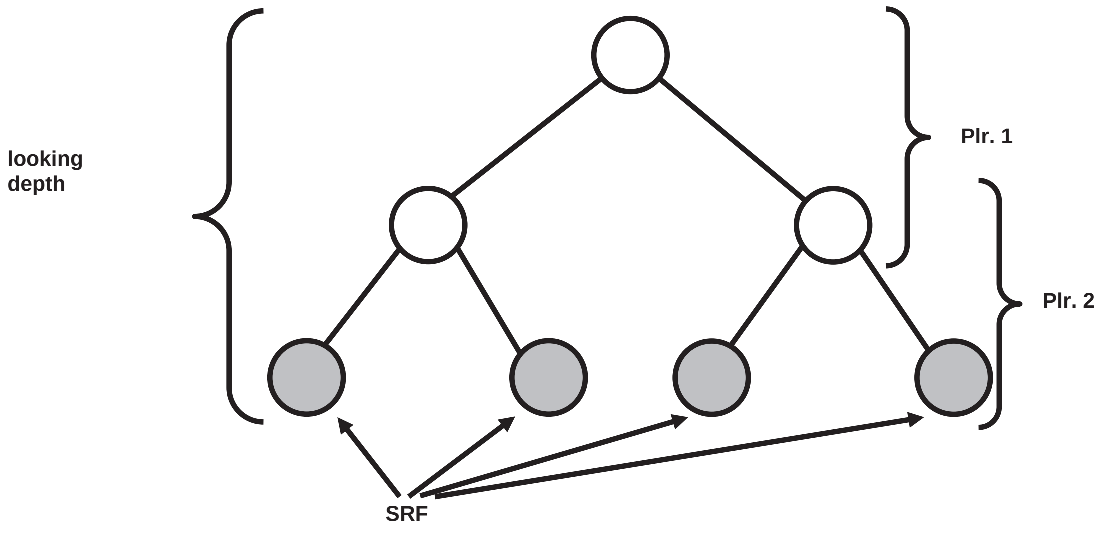

Fig. 6. Score conversion scheme

Alpha-beta pruning algorithm uses branch and bound method for cutting off unnecessary nodes of the tree (see fig. 7). It saves a lot of time.

NegaScout algorithm improves Alpha-beta pruning by using search with a zero window.

There is time control in real games. Each player gets a certain amount of time, e.g. ten minutes. Players are free to decide how to spend time. We implemented time control in our program.

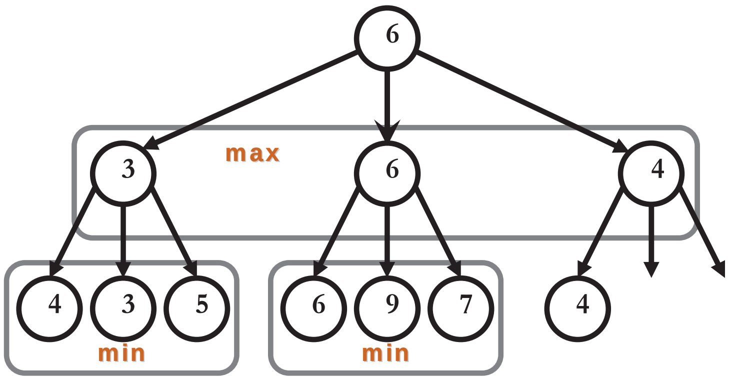

Fig. 7. Alpha-beta pruning algorithm scheme

We set following tasks:

- to check which algorithm is the fastest among the considered;
- to check which time control strategy gives the best results.

We develop the program complex for comparison and testing of the algorithms. It compares and test intelligence of the algorithms and their performance. If time is not limited, all of the algorithms have the same intelligence, but different performance. NegaMax has the worth performance, NegaScout – the best (see fig. 8). If time is limited, NegaScout also has the best intelligence, because it can try a larger number of moves (see fig. 9). For time control we developed some time choosing strategies. It is necessary because of time is limited for a game, not for a one move. The strategies are: simple (uses strictly specific amount of time on the move), defensive (saves time at the end of the game), aggressive (leaves at the end of the game more time).

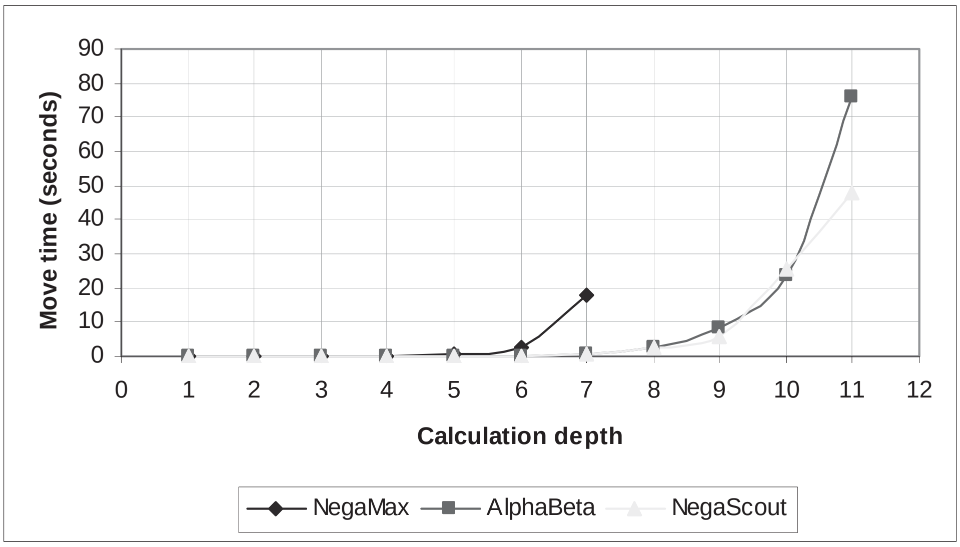

Fig. 8. Algorithms performance testing

Also we tested time control strategies. We conducted thousands of parties for 120 seconds. The defensive strategy was the best because the comparisons were in quick tournaments due to lack of time (it is necessary for many months for a full analysis).

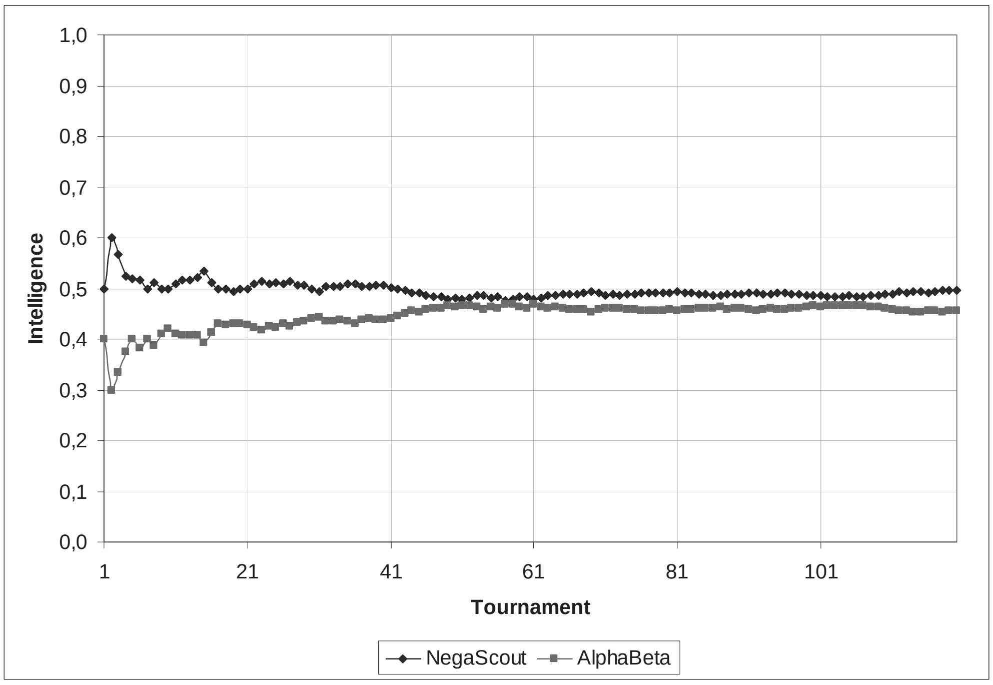

Fig. 9. Algorithms intelligence testing

You can see program complex architecture in the fig. 10.

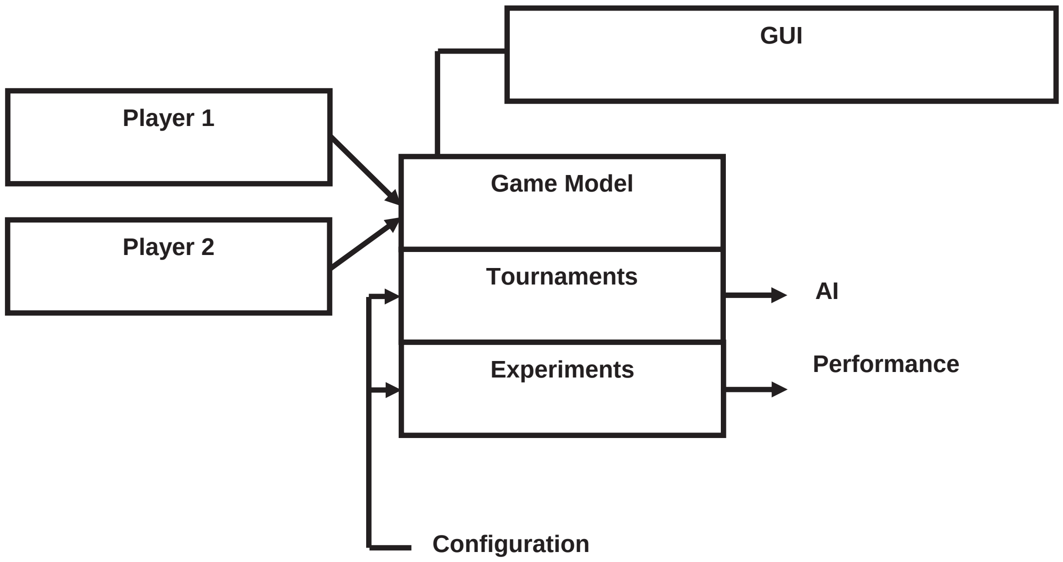

Fig. 10. Program complex scheme

“Experiments” is a special instrument for performance testing. This instrument measures the execution time of the move lot of times. “Tournaments” is a special instrument for intelligence testing. It holds a large number of parties and considers how many times each player has won.

IV. Conclusion

In this work we describe three artificial intelligence algorithms of computer players for the game draughts with multi-threading and time control, implementation details such as graphic user interface and cross-platform working.

There are two main directions of Draughts development:

- Artificial intelligence algorithms improving, adding neural network;
- Time control strategies extension;
- Comparison with well-known draughts engines.

References

[1] Nokia Corporation, “Cross-platform application and UI framework Qt,” *http://qt.nokia.com/products*.
[2] Maemo Community, “Official Maemo OS community web site,” *http://maemo.org*.
[3] MeeGo Community, “Official MeeGo OS community web site,” *http://meego.com*.
[4] Google Corporation, “SVN repository of program,” *http://code.google.com/p/shashki/*.
[5] E. N. Kornilov, “Programming chess and other logic games,” *SPb.: BHV-Peterburg*, 2005, ISBN 5-94157-497-5.