VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

BOARD GAME FOCUSED ON EDUCATIONAL SUPPORT FOR GAMING ALGORITHMS

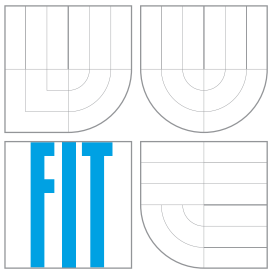
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ČÁSLAVA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DESKOVÁ HRA ZAMĚŘENÁ NA PODPORU VÝUKY HERNÍCH ALGORITMŮ

BOARD GAME FOCUSED ON EDUCATIONAL SUPPORT FOR GAMING ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ČÁSLAVA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. MARTIN DRAHANSKÝ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá oblastí umělé inteligence zvané jako "Metody pro hraní her". Cílem této bakalářské práce je navrhnout a implementovat software, který umožní uživateli snadněji pochopit principy herních algoritmů Minimax a Alfa-beta prořezávání. Typickými uživateli tohoto softwaru mohou být například studenti oboru umělá inteligence. Práci lze rozdělit do dvou hlavních částí. První, teoretická část, obsahuje popis nejrůznějších metod pro řešení úloh a detailněji se zaměřuje na metody pro hraní her. Cílem této části práce je dát čtenáři teoretický základ pro bližší pochopení problematiky herních algoritmů. Druhá část práce je věnována popisu návrhu, implementaci a testování implementovaného softwaru. V závěru druhé části práce jsou shrnuty a diskutovány dosažené výsledky a je zde také nastíněn návrh na možná budoucí vylepšení.

Abstract

This work deals with the part of field of artificial intelligence known as "Methods of playing games". The goal of this bachelor's thesis is to design and implement software that allows the user to more easily understand the principles of game algorithms Minimax and Alpha-beta pruning. Typical users of this software can be, for example, students of artificial intelligence. This work is divided into two main parts. The first theoretical part tries to explain the "Method of playing games" concept and subsequently contains detailed descriptions of software design and educational benefits. The second part of this work is devoted to a description of software implementation, testing and discussion of the achieved results.

Klíčová slova

Umělá inteligence, herní algoritmus, stavový prostor, metoda pro řešení úloh, Minimax, Alfa-beta ořezávání, hra Piškvorky

Keywords

Artificial intelligence, game algorithm, state space, method for task solving methods, Minimax, Alpha-beta pruning, Tic-tac-toe game

Citace

Martin Čáslava: Board game focused on educational support for gaming algorithms, bakalářská práce, Brno, FIT VUT v Brně, 2015

Board game focused on educational support for gaming algorithms

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Martina Drahanského Ph.D. Dále prohlašuji, že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Čáslava
July 26, 2015

Poděkování

Rád bych touto cestou poděkoval mému vedoucímu práce panu doc. Ing. Martinu Drahanskému Ph.D. za drahocenné rady a připomínky, bez kterých by jistě tvorba této práce byla jen stěží možná. Dále bych rád poděkoval kamarádovi Štefanu Martíčkovi za jeho návrhy na zlepšení aplikace a zejména za kritiku, která se podepsala na celkovém výsledku práce.

Contents

Chapter 1

Introduction

This thesis deals with the issues of teaching and presenting the basic principles of methods of playing games. The goal of this thesis is to design, implement and subsequently test the application which, thanks to the game "Tic-tac-toe", allows the user to more easily understand principles of game algorithms (Minimax and Alpha-beta pruning).

Following chapter is devoted to the "Task-solving methods" concept, which is basis for understanding the methods of playing games. The thirth chapter explains the notion "Method of playing games", known as the "Game algorithm" concept, and describes the various types of games on which this thesis is focused. The fourth chapter contains a theoretical design of the application and its graphical user interface and also it is devoted to the description of the educational benefits of the implemented application when compared with similar existing applications. The fifth chapter deals with implementation of the application, description of chosen developmental tools and programming languages. The sixth chapter describes the test scenarios, methods of test execution and achieved tests results. The last chapter is devoted to the discussion of achieved results and subsequently mentions the proposal of possible future improvements.

Chapter 2

Task-solving methods

The aim of this chapter is to give the reader a theoretical base for understanding the methods of playing games. It also tries to explain the notion of "Task-solving methods" and subsequently how this notion is associated with the methods of playing games.

The notion "Task-solving methods" in area of artificial intelligence, is close related with the notion "State space exploration". In essence, it is testing of possible states of the task and identifying what happens in next phases of the task. While playing games almost everyone is considering, in this way, how the opponent would react to his move. [8]
Task-solving methods are evaluated by following criteria:

- **Completeness** - will the method find the solution (if exist)?
- **Time demands** - minimum/maximum/average time required to solve the task.
- **Memory demands** - minimum/maximum/average amount of memory needed to solve the task.
- **Optimality** - will the method find the best solution? [14]

2.1 State space

We can imagine the state space as an oriented graph or tree. For simplifying the terminology is therefore the notion "state space" in the next phases of work, considered as a "tree" (game tree).

Each node of this tree represents a state of task and its edges represent the transitions between them. The route from the initial node (root) to one of its final nodes (leafs) is the solution of the task. Many tasks require a minimization of the route value, which is equal to sum of values of each transition. On the other hand, for some tasks, the route is not important at all and decisive is only the final node. [14]
The state space is defined as the pair:

- **(S,O)**

where:

- **S** - is a non-empty finite set of task states.
- **O** - is a non-empty finite set of operators, which allows to change the states of task. [14]

The task in the state space is defined as the pair:

- **(S_0,G)**

where:

- **S_0** - is an initial state of task.
- **G** - is a set of final states of task. [14]

The solution of the task is defined as a succession of operators:

- $S_1 = O_1(S_0)$, $S_2 = O_2(S_1)$, ..., $S_n = O_n(S_n - 1)$, $S_n \in G$ [5, 14]

For description the state space (image 2.1) is used the following terminology:

- Node **A** - is the root.
- Nodes **I, L, M, ..., Z** - are the leafs.
- Node **C** - is the immediate predecessor of **H** node, etc.
- Nodes **A, D, J** - are the predecessors of **V** node, etc.s
- Node **K** - is the immediate successor of **D** node, etc.
- Nodes **H, I, S, T, U** - are the successors of **C** node, etc.
- Node **A** - has a depth 0, nodes **B, C, D** have a depth 1, etc.
- Node **A** - is the initial state S_0.
- Node **L** - is the final state S_G.
- Expansion od the tree node is the specifying all of its immediate successors.
- Generations of the tree node is its creation.
- Evaluation of the tree node is equal to the sum of transitions from the root to this node. [14]

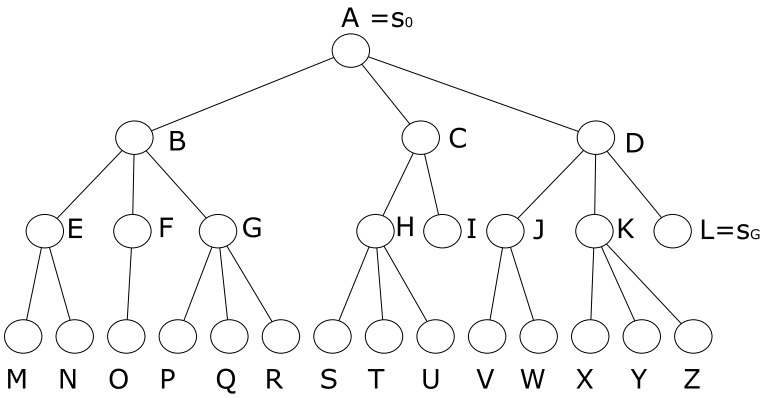


Figure 2.1: State space example [14]

2.2 Methods based on state space exploration

One of the fundamental tasks of artificial intelligence are methods for solving mechanical tasks. Despite high computing power of today's computers, it is for the vast majority of problems unthinkable that a machine is looking for a solution by successive testing of all possibilities. It was necessary to somehow manage the search for the solution.

For these reasons a various methods (algorithms) with different advantages and disadvantages, were invented in recent decades, for exploring the state space. [1]
These algorithms are divided into these following groups:

2.2.1 Uninformed (Blind) Search methods

These methods do not have any information about final state and also do not have any means how to evaluate the current state.

Even people sometimes have to use similar methods - for example when they are searching for the route in the map, from some initial place to some final place, and do not have any clue where the final place is. [14]
Among these methods belong for example:
BFS (Breadth-first search) algorithm
DFS (Depth-first) algorithm
Bidirectional search algorithm

2.2.2 Informed search methods

These methods have an information about the final state and also have the means how to evaluate the current state. Back to the example with searching in the map - if someone is searching for the route in the map from some initial place to some final place, he usually has a rough idea, in which direction from initial place the final place is.

It means if the idea about the location of the final place is more precise, less area of map (state space) is needed to be explored. [14]
Among these methods belong for example:
Beam search algorithm
Greedy search algorithm
A* algorithm

2.2.3 Local search methods

There are some tasks whose solution is only to search for the final state and the route is meaningless. For solving these tasks, methods which instead of searching for the optimal route search for the optimal final state, are used.

These methods are only good for one specific thing, for example for the optimal scatter of the goods on the shelves in the shops etc. [14]
Among these methods belong for example:
Hill-climbing algorithm.

2.2.4 Methods for decomposition to subtasks (AND/OR graphs)

Decomposition to subtasks is possible to typify by graph (tree) as in other methods. The difference is that the nodes do not represent the states of task, but subtask. Each following

node (subtask) can be expanded to easier subtask, until the leafs (final nodes) do not correspond to the elementary tasks, or unsolvable tasks. The other difference is that the nodes can acquire only the boolean types "AND" or "OR". [11]

- **OR problem** - the task A (image 2.2) is soluble, if there is at least one of its subtasks soluble (tasks A, C, D). [14]
- **AND problem** - the task E (image 2.2) is soluble, if there are all of its subtasks soluble (tasks F, G, H). [14]

Among these methods belong for example:
AO algorithm

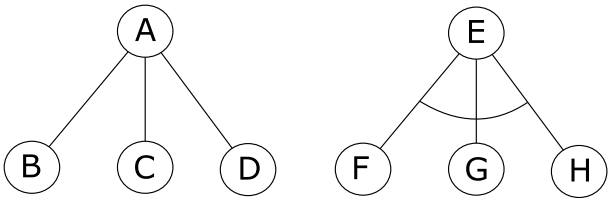


Figure 2.2: AND, OR problems [14]

For more information about the algorithms mentioned in this chapter, you can use following link: https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-5a.pdf

Chapter 3

Methods of playing games - game algorithms

This work takes into account the games for two regularly alternating players. Both of these players have a complete idea about the state of game and each player is trying to win. The problem lies in finding the optimal move for the player on turn (player A). Because the next move is the opponent's move (player B), every move which leads to the victory of the player A, has to be unsolvable for the player B, in different words, all of the player B moves, have to be soluble for player A (AND problem).

Searching for the move which leads to the victory, leads to the exploration of the AND/OR graph 2.2.4. After selection and execution of the optimal move of the player A „everything is forgotten", in the next turn player B is playing, and player A chooses his move from the new state of game again. [14].

Thus described games can be divided into following categories:

3.1 Primitive games

For this kind of games, it is possible to explore the whole AND/OR graph in real time. In case of finding the solution of the game, it is not necessary to return the whole part of graph, but only the move of player A, which leads to his victory.[14]

As the example of primitive game, serves the following example of the „Tower of Hanoi" game. 3.1

3.1.1 Tower of Hanoi game

In the initial state of the task is the Tower of Hanoi consisting of N disks of different diameters, situated on left pin (A pin). The goal of task is to move the disks to the right pin (pin C) using the pin in the middle (pin B).

it is only allowed to move the upper disk and the disk must not be placed on the disc of smaller diameter. Pins are denoted by diameter as the integers 1, 2, ..., N.[11]

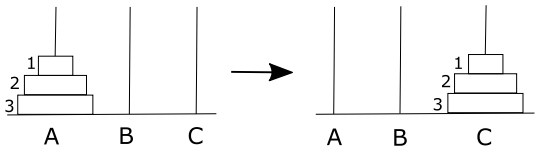


Figure 3.1: Tower of Hanoi - game example [11]

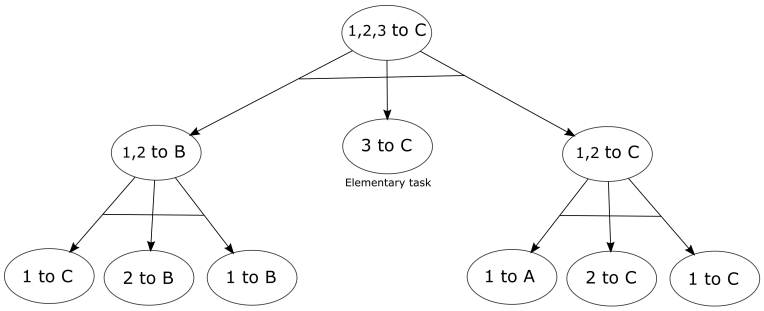


Figure 3.2: Tower of Hanoi - decomposition to subtasks [11]

3.2 Difficult games

In these games, the game tree is explored only to a predetermined depth. If at this depth are not the nodes, for which is possible to decide about task solvability or insolvability, it is necessary to evaluate the tree nodes somehow.

In these methods the evaluation function is used for valuating the tree nodes. The positive values indicate the favorable conditions for player A (the bigger, the more favorable), the negative values indicate the positive status for player B (the the smaller, the more favorable). Winning or losing is assessed as the maximum of these numerical values of the considered interval (for example 1, 0, -1). It is obvious that player A selects the moves leading to nodes with the maximum valuations, and player B chooses moves leading to the nodes with the minimum valuations. The basic game algorithm works on that principle and is therefore called the Minimax. [14]

3.2.1 Minimax

These methods work on the principle of exploration of the game tree with restrictions of its depth. For this algorithm, the static evaluation function *f*, which evaluates each tree node on the i level, is specific. This evaluation function works on the principle of the following iterative algorithm: [13]

- Tree node is expanded and for all its successors is determined the value of *f*. [13]
- From these determined values the best value is selected. This value is reversely used as the evaluation of the parent node at the i level. [13]

Minimax algorithm expects the restrictively allowed depth of exploration of the tree. For the effectiveness of this algorithm is the "best" value of the evaluation function *f*, the deciding factor.

As mentioned in the chapter 3.2, if it is player A's turn, as the best value is considered the **maximum** value of the function f at the closest lower level. Conversely if it is player

B's turn, as the best value is considered the **minimum** value of the function *f*, at the the closest lower level. It is therefore logical that player A is trying to maximize the profit of player B, and player B is trying to minimize the profit of player A. [13]

Principle of the Minimax

Suppose the situation at the image 3.3. If the tree is explored to the depth 1, the node A is evaluated as the maximum value of the evaluation function *f*, of its successors (B, C, D). During the reverse evaluation process, the nodes B, C, D in the second level, are evaluated as the minimum value of the evaluation function f of its successors at the third level. [13]

- $f(B) = min\{f(E), f(F), f(G)\}$ [13]
- $f(C) = min\{f(H), f(I), f(J)\}$ [13]
- $f(D) = min\{f(K), f(L)\}$ [13]

At the higher level is the node A evaluated as the maximum of its successors (B, C, D).

- $f(A) = max\{f(B), f(C), f(D)\}$ [13]

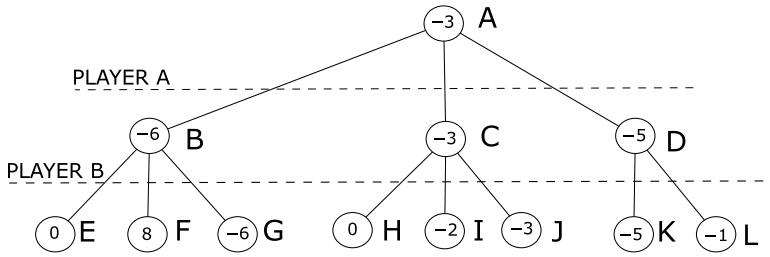


Figure 3.3: Minimax algorithm principle

Pseudocode of the Minimax

```
int score;
int optimal_opponent_move;
check the state of game and evaluate the tree node;
for all (empty game positions)
{
   if(player == MIN_PLAYER)
   {
      take the game position;
      score = minimax(MAX_PLAYER, depth-1, bestMaxScore, bestMinScore);
      free the game position;
      if(score < bestMinScore)
      {
         bestMinScore = score;
         if (depth == 0)
            optimal_opponent_move = position;
      }
```

```
   }
   else if (player == MAX_PLAYER)
   {
      take the game position;
      score = minimax(MIN_PLAYER, depth-1, bestMaxScore, bestMinScore);
      free the game position;
      if(score > bestMaxScore)
      {
         bestMaxScore = score;
      }
   }
}
if (player == MAX)
   return bestMaxScore;
else
   return bestMinScore;
```

Complexity

The algorithm has very low memory demands, because it does not need to remember the whole section of the tree, when calculating. Only the current path from the root to the leaf and immediately following moves is saved in the memory.

The problem is the exponential time complexity. In case of the tree with constant branching factor x and the depth y is the time complexity x^y. The calculation of the time complexity shows the weakness: for the games that have a large branching factor, this algorithm can not be effectively deployed in greater depth of exploration. In practice it is therefore prefered to use the algorithms derived from Alpha-beta pruning, which achieves, compared with the Minimax, almost twice larger depth of exploration, in the same time. [2]

3.2.2 Alpha-beta pruning

This algorithm is based on the principle Minimax algorithm 3.2.1, but it is improved by technique (*branch-and-bound*), which allows to decide, whether the next branch of the tree is useless to explore, or not.

In case that exploration of some branch is useless, the branch is cut off and is not explored. This technique allows in the very early stages of the tree exploration, to reject the solution which is evidently worse than already found solutions. Thanks to this technique there is no need to explore the whole tree, but only its „interesting“ parts.[13]

Principle of the Alpha-beta pruning

Unlike the Minimax, the Alpha-beta pruning uses besides the integer value of the node, other two values α and β.

- $\boldsymbol{\alpha}$ - this value represents the lower limit of the evaluation of the tree node, corresponding to the move of the player A.
- $\boldsymbol{\beta}$ - this value represents the upper limit of the evaluation of the tree node, corrispinding to the move of the player B.

On the basis of these values, the algorithm decides whether the branch of tree will be cut off or not. The α value is calculated on the level of player A, as the maximum value of successors of the current node and the value of the α from the parent level. On the level of player B, the value of α does not change. Analogously, the β value is calculated on the level of player B, as the minimum value of successors of the current node and the value of the β from the parent level.

Values of α and β are not therefore global minimum or maximum values, but the „bubbling“ values, between parts of the tree. The cutting of the branch may occur at any level. At the level of player A, the α cuts may occur and at the level of player B the β cuts may occur. [10]

- $\boldsymbol{\alpha}$ **cuts** - the cutting and stopping exploration of the next branch of the tree occurs, when during the reverse evaluation process, as in the Minimax algorithm, is fulfilled the condition: $\boldsymbol{\alpha \geq \beta}$
- $\boldsymbol{\beta}$ **cuts** - the cutting and stopping exploration of the next branch of the tree occurs, when during the reverse evaluation process, as in the Minimax algorithm, is fulfilled the condition: $\boldsymbol{\beta \leq \alpha}$

At the beginning of the algorithm the values alpha and beta are initialized on:

- $\boldsymbol{\alpha = -\infty}$
- $\boldsymbol{\beta = \infty}$

The image 3.4 shows the principle of Alpha-beta pruning algorithm. The red nodes denotes the parts of the tree which the algorithm did not explore (cuts).

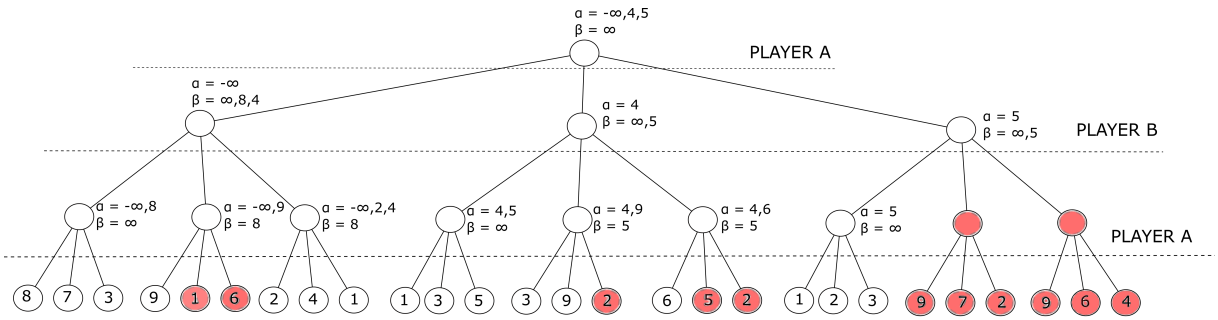


Figure 3.4: Alpha-beta pruning algorithm principle [14]

Pseudocode of the Alpha-beta pruning

```
int score;
int optimal_opponent_move;
check the state of game and evaluate the tree node;
for all (empty game positions)
{
   if(player == MIN_PLAYER)
   {
      take the game position;
      score = minimax(MAX_PLAYER, depth-1, alpha, beta);
      free the game position;
      if(score <= alpha)
         cut the rest of the tree nodes in this branch;
      if(score < beta)
      {
         beta = score;
         if (depth == 0)
            optimal_opponent_move = position;
      }
   }
   else if (player == MAX_PLAYER)
   {
      take the game position;
      score = minimax(MIN_PLAYER, depth-1, alpha, beta);
      free the game position;
      if(score >= beta)
         cut the rest of the tree nodes in this branch;
      if(score > alpha)
      {
         alpha = score;
      }
   }
}
if (player == MAX)
   return alpha;
else
   return beta;
```

Complexity

For maximum efficiency of this algorithm it is suitable to use some heuristics for sorting the game moves. The cutting of the nodes is more effective, when the exploration of the moves is carried out in the right order. When the game moves are in the optimal order, the time complexity of Alpha-beta pruning is $x^{y/2}$, which means, that in the optimal case, this algorithm can reach twice the depth of exploration, of the Minimax, in the same time.

It is possible to prove, that in case of incorrect selection order of the game moves, the algorithm can reach the time complexity x^y, which is the time complexity of the Minimax algorithm. [9]

3.3 Games with uncertainty

There are many such games, for two regularly alternating players where both of them have complete information about the state of the game. They play honestly and both of them want to win.

Unlike the above mentioned methods, when playing these games, the players need to use a dice and thus the uncertainty enters the game. The basic principle of the games with dice is described in the image 3.5. The player A is on the turn and just threw the dice (considered is the classic six-party dice). The result of the throw is No. 4, however, this fact is not important for further consideration. [14]

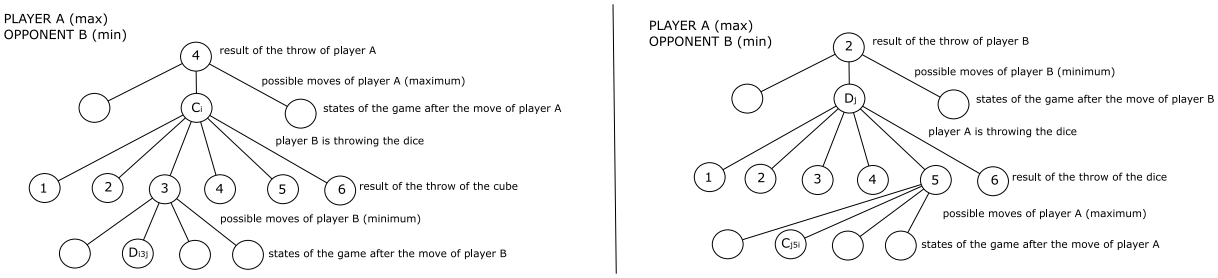


Figure 3.5: The principle of the games with uncertainty [14]

Player A knows, which game moves C_i he can realize. Of course he can evaluate the individual states of game (immediate successors) and choose the move leading to the state with the maximum value. While a lot of people use this approach, it is certainly not interesting. Player A, will therefore proceed with the evaluation of each state C_i in more complicated way (the left part of the image 3.5).

Working on the assumption, that player B, would for the known result of the throw, choose the move to the state D_j, with the minimum value. However player B does not know the result of his throw, so he can work only with the expected value, *expectimin* (**expected minimum**). [14]

$$expectimin(C_i) = \sum_k P(h_k) * \min_j (D_{jki}) \tag{3.1}$$

The eaquation 3.1 is taken from [14] and serves for calculating the **expected minimum**. In this equation is:

- h_k - the k-th result of the throw (1, 2, 3, 4, 5, or 6).
- $P(h_k)$ - the probability of the k-th result (for games with one dice, the probability for all of the results is the same: $P(hk) = 1/6$, for the games with two dice, with the same numbers at the dice, the probabilities of the results are 1/36 and the probabilities of the results, with different numbers at the dice, are 1/18).
- D_{jki} - the evaluation of the state D_j, which is reachable from the state C_ii, after the k-th result of the throw of the dice.

The *expectimin* is therefore given by the sum of all the possible values of the results, after the player has thrown the dice. Each value is given by the product of the probability of the result of the throw and subsequent the minimum evaluation of the state, which is possible to reach after the throw. Player A then chooses the move to the state C_i with the maximum

value *expectmin*. A similar procedure is followed in the investigation of the expected value of the node D_{ijk} (which is in the right part of the picture 3.5, for simplicity referred as the D_j). Because player A chooses the maximum of the possible values, this evaluation is denoted as *expectimax* (**expected maximum**). [14]

$$expectimax(D_j) = \sum_k P(h_k) * \max_j (C_{jki}) \tag{3.2}$$

The eaquation 3.2 is taken from [14] and serves for calculating the **expected maximum**. In this equation is:

- h_k - the k-th result of throw (1, 2, 3, 4, 5, or 6).
- $P(h_k)$ - the probability of the k-th result ...
- C_{jki} - the evaluation of the state C_i, which is reachable from the state D_{ji}, after the k-th result of the throw of the dice.

Chapter 4

Application design

As the name of the work suggests, the goal of this thesis is to create an application, which thanks to the board game, allows to demonstrate the principles of game algorithms Minimax and Alpha-beta pruning.

For this reason, I decided to implement the game called as "Tic-tac-toe". This game is well known, has a simple rules and one game does not take a lot of time. Whole application is conceived as the educational tool. The user has an option to choose one of two implemented game algorithms as his opponent and watch step-by step how the algorithms work.

4.1 Tic-tac-toe game

The Tic-tac-toe is the strategy game for two regularly alternating players. This game is played on the squared paper. Both players are alternating in the drawing the game marks (crosses or wheels). The winner is the first player to place his five game marks in the orthogonal or the diagonal direction.

For the applications purposes the game rules were a little bit changed. The game desk has size 3×3 game squares and the winner is the first player to place his three game marks in the orthogonal or the diagonal direction.

Figure 4.1: Tic-tac-toe example [6]

4.2 Graphics user interface

When designing the graphics user interface (in next phases of the work this notion is named as GUI) the emphasis was on maximum simplicity and intuitiveness of its use. The application control system is desribed in chapter 4.4.

The GUI comprises of one main window, two buttons for controlling the application and the simple list menu on the top left of the main window. The main application window is divided into several parts:

- **Game board** - this part of GUI represents the game board for the game "Tic-tac-toe". Thanks to the clicking the right mouse button into the squares of game board, the user can place his game marks, and play the game. If the user does not choose any

game algorithm as his opponent, it is possible to play the game against the another human player. But this effect is rather secondary.

- **Window for displaying the application output** - this window serves for displaying the application output, which is represented as the corresponding game tree. Each node of this tree represents the one state of the game. Thanks to the special **next step** button, the user has an option to simulate the game algorithm activity step-by step. In the image 4.2 you can see the application screenshot with fully explored game tree.
- **Window for reading the pdf files** - in this window, the user can read detailed information about implemented game algorithms. This window also serves for displaying the help.
- **Window for listing the game algorithm details** - this side panel serves for displaying detailed information regarding the implementation of the chosen game algorithm. In each game step, the user is able to see, what is happening in the background of the selected game algorithm. For example: which conditions are evaluated, which players is on the turn, which values is the algorithm selecting etc.

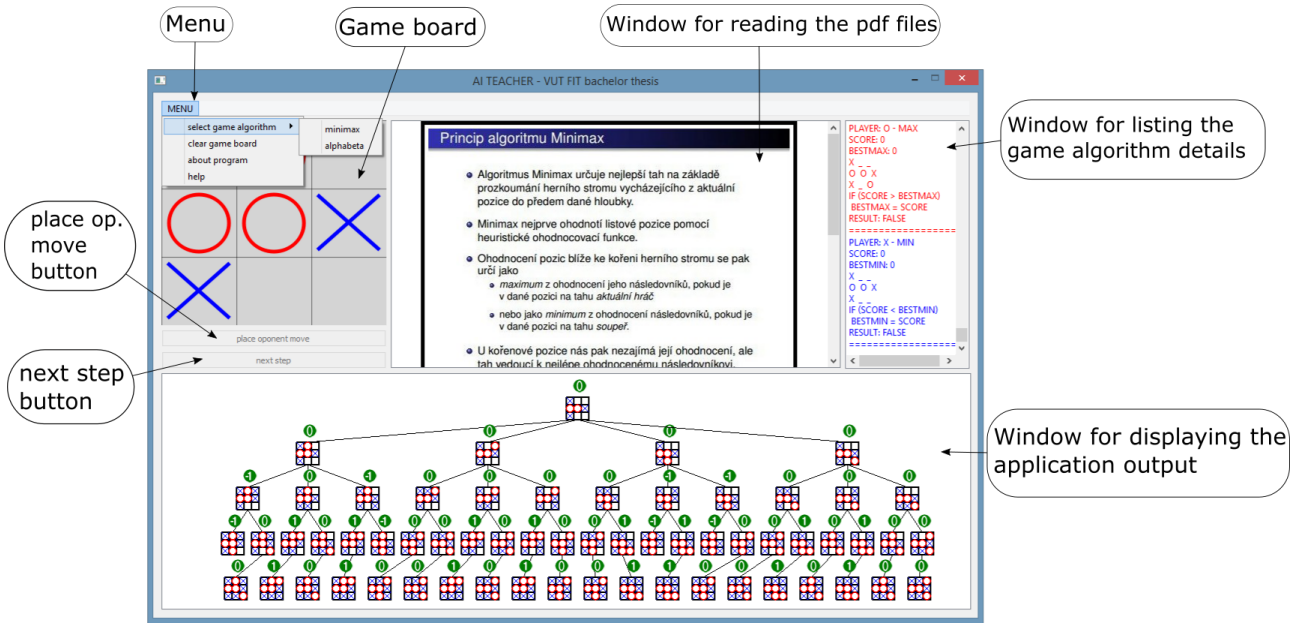


Figure 4.2: Application screenshot - GUI

4.3 Application object model

When designing the application, it was more than clear that it is needed to use the object oriented approach and design the application object model. The object model is divided to seven main classes and one data structure:

- **Class: Square** - this class represents one square in the game board for the Tic-tac-toe game.

- **Class: Board** - this class is inherited of the **Square** class and represents the whole game board for the Tic-tac-toe game.
- **Class: Empty_node** - represents one non-expanded tree node.
- **Class: Tree_node** - represents one expanded tree node.
- **Class: Tree_node_data** - contains the information about tree nodes.
- **Class: Check_game_status** - includes an auxiliary methods for artificial intelligence.
- **Class: Artificial_intelligence** - contains the implementation of the game algorithms Minimax and Alpha-beta pruning.
- **Structure: Game_data** - contains important control variables, pointers to the other objects etc. Each object in this object system has permission to read the data from this structure, some of them even to write.

The image 4.3 shows the application object model and the way the classes are communicating among themselves. The implementation of this object model is described in the chapter 5.1.

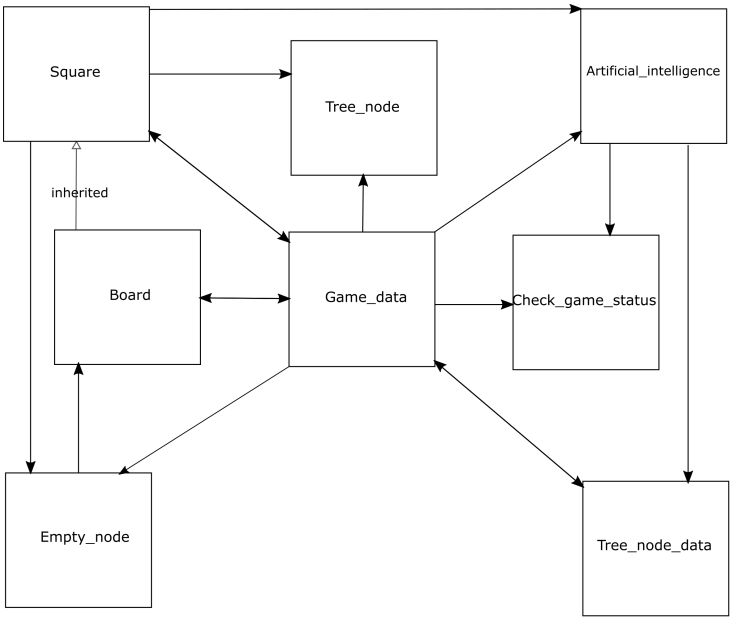


Figure 4.3: Application object model

4.4 Application control system

The application control system was designed with respect to the maximum simplicity and its intuitiveness. After starting the application, the mandatory user input is expected, in form of the starting game situation and selection of the game algorithm. After entering the valid user input, the application generates the output in form of the coresponding empty game tree. The application output can be further modified by clicking the appropriate

next step button, and simulate the activity of the chosen game algorithm. The image 4.4 shows the model of application control system.

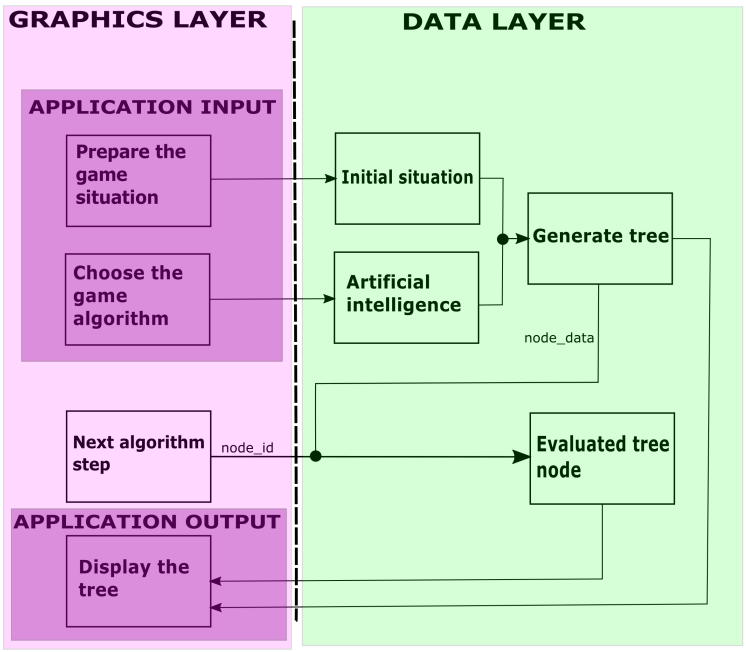


Figure 4.4: Application control system

4.4.1 Application user input

The entering of the user input data, proceeds in two mandatory steps in this order:

- **Preparing the starting game situation** - the user, thanks to the clicking the right mouse button at the game fields, places the game marks and prepares the starting game situation. This situation is the initial state for the game algorithm and concurrently is the root of the generated game tree. The condition is that the last placed game mark must belong to the user (cross mark), because the next player on the turn must be the opponent (circle).

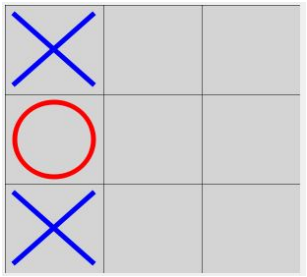


Figure 4.5: Example of the possible starting game situation (the user plays the cross mark)

If the user prepared the incorrect starting game situation, he is reminded by the special warning window.

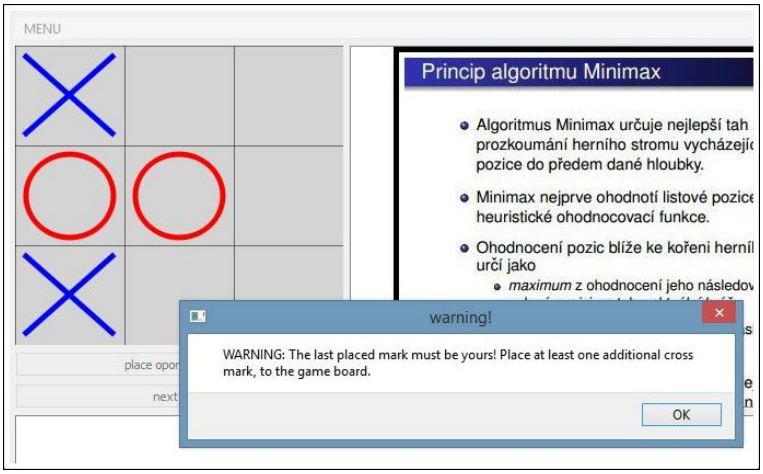


Figure 4.6: Reaction for the incorrect user input.

- **Choosing the game algorithm** - after preparing the starting game situation, the user is expected to choose one of two implemented game algorithms.

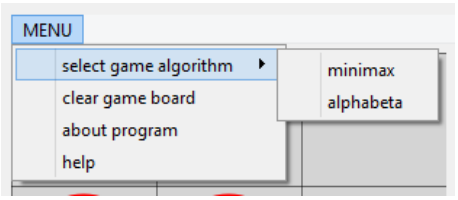


Figure 4.7: Choice of the game algorithm

4.4.2 Application output

After entering the correct input data, the user can run the generation of the application output. The standard application output is represented as the empty unexplored game tree (image 4.8). This tree is possible to be modified and simulate the activity of the chosen game algorithm (images 4.9, 4.10).

The simulation of the game algorithm activity, proceeds in the following manner. After clicking the **next step** button, one expanded tree node is displayed. Above each expanded node is displayed its relevant value, and the bitmap of the arrow. This arrow gives the user accurate idea, in which part of the tree is the algorithm located. Thanks to the **next step** button the user can see step-by step, the recursive plunging of the chosen game algorithm and the process of evaluating of the tree nodes. After expanding all tree nodes, the user has an option using the **place opponent move** button to find out, which game move the algorithm chose as its optimal move.

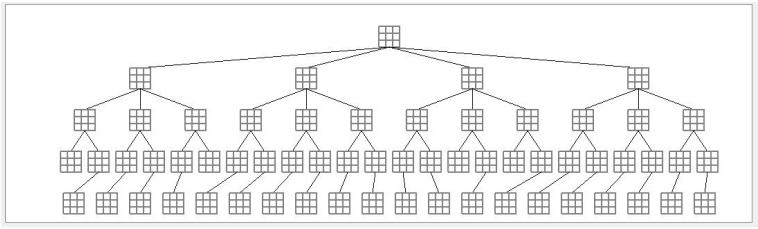


Figure 4.8: Empty unexplored game tree

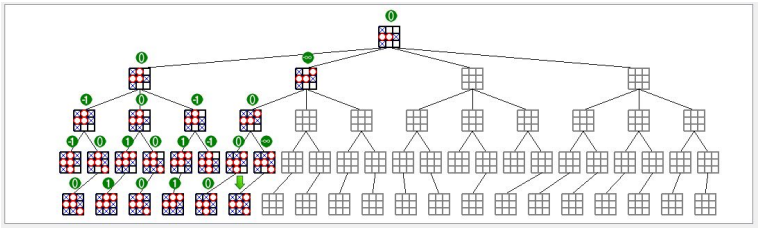


Figure 4.9: Partly explored game tree by Minimax algorithm

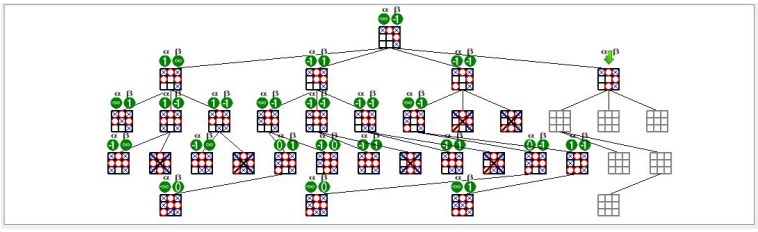


Figure 4.10: Partly explored game tree by Alpha-beta pruning algorithm

4.5 Educational benefits

During the software development process, every software developer should be able to answer the question: "*Why would the user want to use this software?*". This question, should be even more emphasized, when the similar software already exists.

In this chapter, the main advantages of the developed application are mentioned, as well as the similar existing applications and their comparison. The disadvantages, and their possible future improvements are mentioned in the chapter 7.1. Because the developed application is conceived as the teaching tool, its advantages are to be the educational benefits. Mentioned benefits in this chapter, are considered the subjective benefits from the developer's perspective. The chapter 6 contains the research, whether these proposed benefits are real benefits for the users.

4.5.1 Benefits

It is hard to imagine, that the users will want to use this software in a long time period. In most cases, the aim of its users, will be to run the application, understand the game algorithm principles and not to run the application anymore.

For that reason, the proposed benefits are aimed at the maximum effectiveness of understanding the issue, in the shortest possible time. The main application benefits are:

- **Connection between the algorithms principles and the game** - majority of the similar existing applications tries to explain the game algorithm principles on the example with game tree, where the tree nodes are represented as the empty circles, and the user has only the option to watch, how the values of the tree nodes (circles) are changing. Since as these algorithms are primary developed for playing the games, I decided to implement the simple game Tic-tac-toe 4.1 and connect the playing of this game, with the demonstration of the game algorithms principles. This connection should give the user clearer idea how these algorithms work.

- **Detailed description, of how the algorithms work, on the level of implementation** - thanks to the special window, which is described in chapter 4.2, the user can see, what is happening in the each algorithm step, on the level of the source code. The advantage is, that the user not only understands the algorithm principles, but also gets the idea how the algorithm is implemented, or eventually how to implement it.

4.5.2 Similar existing applications

Vast majority of the similar applications run online. For running these applications, in most cases, the user just needs an internet connection, which is nowadays negligible problem, and the web browser with installed Java plugin. Indisputable advantage of these applications, is that the user does not have to install any software on his hard drive.

On the other hand, the problem is, that the creation of the user friendly GUI for the web applications is a bit difficult, and most of these applications with slight differences, look the same, as you can see in the following images 4.11, 4.12.

Links to similar web applications:
http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html
http://kra.lc/projects/gamevisual/launch.php

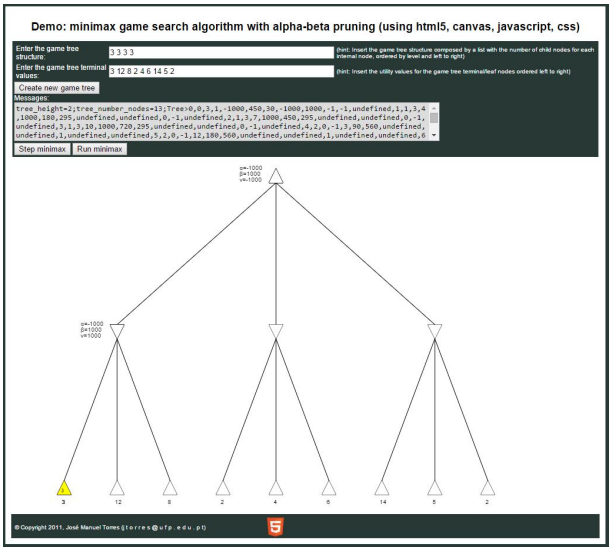


Figure 4.11: Example of similar existing application [12]

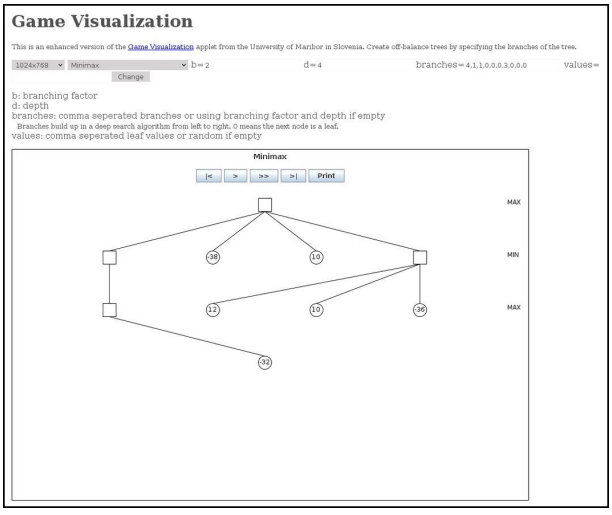


Figure 4.12: Example of similar existing application [7]

Chapter 5

Application implementation

This chapter describes the implementation of the most important classes, presented in the chapter 4.3. For implementation of the application, the programming language *C++* version 10 was used. The *QT* framework version 3.2.0 was used for creation of the GUI.

5.1 Implementation of the object model

The application object model is divided into 7 main classes:

Class Square

The instance of this class, represents one field (square) of the game desk for the Tic-tac-toe game. Each object of this class has an information, which game mark is placed on it, and also the information about its position on the game desk. For holding the information about the game mark, serves the instance variable `int type` and for holding the information about the square position, serves the instance variable `int position`.

The instance variable `type` can reach the integer values $0, 1, 2$. The zero value means that the square is empty, and the player is able to place his game mark at this square, the value 1 means that at the square is already placed the circle mark and the value 2 means, that square is already occupied by the cross mark. The instance variable `position` can reach the integer values $1, 2, 3, 4, 5, 6, 7, 8, 9$. In the whole object system the instance of this class exists 9 times.

Instance methods:

- `void generateTree()` - method for generating the empty, uexplored game tree, which is displayed on the graphics output. Each node of this tree, is in the default situation presented as the empty grid. You can see the example of this tree in the image 4.8.
- `static void oponentMove(square *s)` - this method shows the optimal opponent game move, after the end of run of the game algorithm.
- `static void insert_node_into_view(int x, int y, int id)` - this method allows to create and insert the one expanded tree node, in the graphic form, into the

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing the facilities for low-level memory manipulation. [4]

Software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. [3]

graphics output. The first two parameters `x` and `y` are the coordinates on which the node will be inserted. The last parameter `id` is the identificator of the inserted node. Each of these nodes is default set as the invisible. If the user wants to modify the output, the node will become visible. The pointers to these objects are stored in the array of pointers `tree[]`, which is saved in the `game_data` structure.

- `void paintEvent(QPaintEvent *)` - this method serves for the displaying the game marks, after the click, on the free game field in the game board. Which mark will be shown, depends on the instance variable `type`.

- `void mousePressEvent(QMouseEvent *e)` - method for handling the action, after the click on the square object. In case of click on the empty square, to the variable `type` is assigned the value 0 or 1, according to, which player's turn it is. After assigning the value to the variable `type`, the method `paintevent` is invoked and then the information about the mark is saved, on the currently clicked square. If the user chose the artificial intelligence as his opponent, the method `minimax`, or `alphabeta` is invoked from the class `Artificial_intelligence` and then the method `generateTree` is invoked. In the end comes the other player's turn.

Class Board

This class is inherited of the `Square` class and represents the whole game board for the Tic-tac-toe game. The current data configuration of this board, is saved in the array of 9 characters, `game_desk[]`.

This array is saved in the data structure `game_data`. Each character in this array can reach values "_, X, O". The value "_" means, that the game field is empty, the values "X, O" mean, that the position is already occupied by X, or O player. At the beginning of game, every character in this array is initialized to the "_" value.

Instance methods:

- `void minimax_selected()` - after choosing the Minimax algorithm from the application menu, this method sets the `ai_menu` flag to the 1. When the `ai_menu` is set to the value 1, it means that the Minimax algorithm was selected, value 2 means the Alpha-beta-pruning was selected and 0 means that no game algorithm was selected.

- `void alphabeta_selected()` - after choosing the Alpha-beta pruning algorithm from the application menu, this method sets the `ai_menu` flag to the 2.

- `void restart()` - this method initializes all important variables to their initial values. The values of the `game_desk[]` are also initialized to the "_" values.

- `static void showNode()` - this method serves for showing the expanded tree nodes. When the user wants to modify the application graphics output and simulate the activity of the chosen game algorithm, after each click on the **next step** button, the expanded game node is set as **visible**. The pointers to these expanded nodes, are saved in the array `tree[]`. This array is also saved in the `game_data` structure. This method also shows the bitmaps of arrows, above the every expanded node. Thanks to this arrow, the user exactly knows, which node of the tree, the game algorithm goes through. The pointers to these bitmaps are saved in the array `bitmap_arrow[]`. This array is also saved in the `game_data` structure.

- `void help()` - method for displaying the application help.
- `static void printNodeDetail()` - this method serves for displaying the detailed information, about what is happening during the each game algorithm step. This information is shown in the window for listing the game algorithms details, which is described in the chapter 4.2.
- `void onDetailItemClicked(QListWidgetItem *item)` - this method handles the activity when the user clicks on the listing about the game algorithms details.
- `void blickNode()` - when the user clicked on the listing about the game algorithm details, the node which is related to this listing, the user clicked on, starts flickering.

Class Empty_node

This class represents one **non-expanded** tree node in graphics form, which is displayed as the standard graphic output, which is described in the chapter 4.4.2
Instance methods:

- `QRectF boundingRect() const)` - method which returns the outline of the the tree node.
- `void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event)` - this method is invoked, when the user clicks twice on some non-expanded node in the graphics output. After double clicking on the arbitrary non-expanded node, the part of the tree is automatically explored, to the node which the user clicked.
- `void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget)` - this method creates the grid (miniature of the game desk for the Tic-tac-toe game) from the outline, which returns the `boundingRect` method.

Class Tree_node

This class represents one **expanded** tree node in graphics form, which is displayed as the modified graphic output, which is described in the chapter 4.4.2
Instance methods:

- `QRectF boundingRect() const` - method which returns the outline of the the tree node.
- `void paint(QPainter *painter,const QStyleOptionGraphicsItem *option, QWidget *widget)` - this method creates the grid (miniature of the game desk for the Tic-tac-toe game) from the outline, which returns the `boundingRect` method.
- `void drawCross(QPainter *painter)` - method for drawing the cross mark into the outline of the tree node.
- `void drawCircle(QPainter *painter)` - method for drawing the circle mark into the outline of the tree node.

Class Tree_node_data

Instance of this class, represents one tree node in its data form. Each instance of this class contains following instance variables: `char node_game_desk[]` is the array, which holds the information about the configuration of the node.

The `int id` is the unique identificator of the node. The `int father` is the identifckator of the predecessor of this node. The `level` is the depth of the node. The `int end_of_game` is the value of this node (0, 1, -1). The `int alpha` and `int beta` are the values of α and β in case the user chose the Alpha-beta pruning algorithm. The `bool cut` means that the node is cut off.

Instance methods:

- `static void add_node_into_tree(int depth, int father)` - this method creates the node in the data form and inserts it into the tree. The first parameter `depth` is the depth, where the node is created, the second parameter `father` is the identifikator of its predecessor.

Class Check_game_status

This class checks, whether in the tree node occured the end of game. In case the end of game occurred, methods of this class return the value of the node.

The value 0 is returned in case of draw, the value 1 is returned in case of victory of player X and the value -1 is returned in case of victory of player O. In case, the end of game does not occur, the initial values of the tree nodes are set to ∞ and $-\infty$.

Instance Methods:

- `static int checkDraw()` - method which checks whether during the playing game occurred the draw.
- `static int checkWin()` - method which checks whether during the playing game occurred the victory of some player.

Class Artificial_intelligence

This class implements the game algorithms Minimax and Alpha-beta pruning.

Instance methods:

- `static int minimax(int player,int depth,int bestMaxScore,int bestMinScore)` - this method implements the Minimax algorithm. The first parameter `player` is the player on the turn, second parameter `depth` is the depth of the recursion and two last parameters `bestMaxScore` and `bestMinScore` are the best score of players X and O.
- `static int alphabeta(int player, int depth, int alpha, int beta)` - this method implemets the Alpha-beta pruning algorithm.

Structure Game_data

Data structure, which contains very important control variables, pointers to the other objects etc. Each object in this object system has permission to read the data from this structure, some of them even to write.

The most important variables in this structure are: `node_data *tree[]` which is the array of pointers to the instances of the `Tree_node_data` class, which represents one tree

node in its data form. The `tree_node *visible_node[]` which is the array of pointers to the instances of the `Tree_node` class, which represents one tree node in its graphic form. The `char game_desk[]`, this array represents the game board for the Tic-tac-toe game in the data form.

The last important variable in this data structure is the `QGraphicsScene *TreeViewScene`, which is the pointer to the graphic application output.

Chapter 6

Application testing

This chapter is dedicated to the application testing, describes the test scenarios and achieved test results. The application was tested in two independent phases. In each phase, played the important role its potential users.

The first phase, is focused on testing the application functionality. The second phase, is focused on testing the usability of the application for the real users. The application was tested on the operational systems *Windows 8.1 x 64* and *Fedora linux 10*.

6.1 Testing of the application functionality

In the first phase of testing, was tested, whether the implemented algorithms work correctly and whether the application submits the valid graphics outputs. The validity of the application output was tested in following manner.

I implemented the algorithms Minimax and Alpha-beta pruning as the simple, independent test application, which runs in the command line. The input of this application is the initial game situation in the text form. The output of this application is the text file, containing the informations about the generated tree 6.1. Then this output was compared with the real graphic output of the main application.

When the new application functionality was implemented, the application in form of the prototype, was submitted to the real users. During the implementation process, the users already had a choice to try the application. The users were asked, to try the application in various, unexpected situations, which came to their mind. This kind of testing led to the finding of many application glitches, which from the developer's perspective were difficult to discover. Another important aspect is, that the release of these prototypes, allowed to react to the users requirements, and improve the application functionality, during the implementation process.

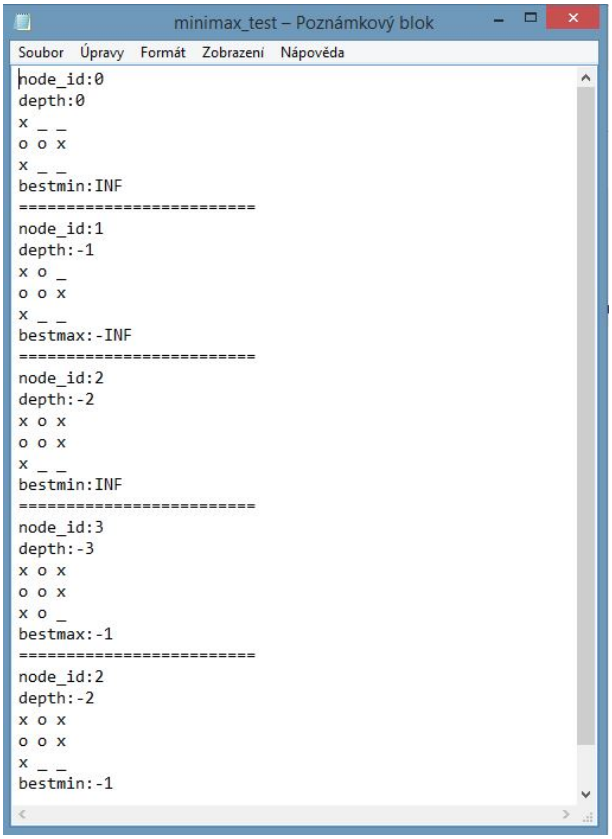


Figure 6.1: Example of the test application output

6.2 Testing of the real use

After finishing the implementation process, the aspects of the real use of the implemented application were tested. The whole process of testing was divided into the four simple sub-tests.

The reason I decided to divide the test into sub-tests, was that it might be demotivating for the users, to solve the difficult tasks at start. These sub-tests were not so difficult to solve and the completion of each sub-test was motivating for the users to learn, how the game algorithms work. First test, was focused on controlling the application. The next three tests covered the learning process of the game algorithms principles.

This testing was executed on the sample of 7 real international users, each of these users is the student of the IT university.

Sub-test1 scenario: GUI control

The users were familiar with the GUI and their task was to prepare random initial game situation, run the arbitrary game algorithm and then generate the game tree.

In the diagram 6.2 you can see the times, which the individual users needed for solving the task.

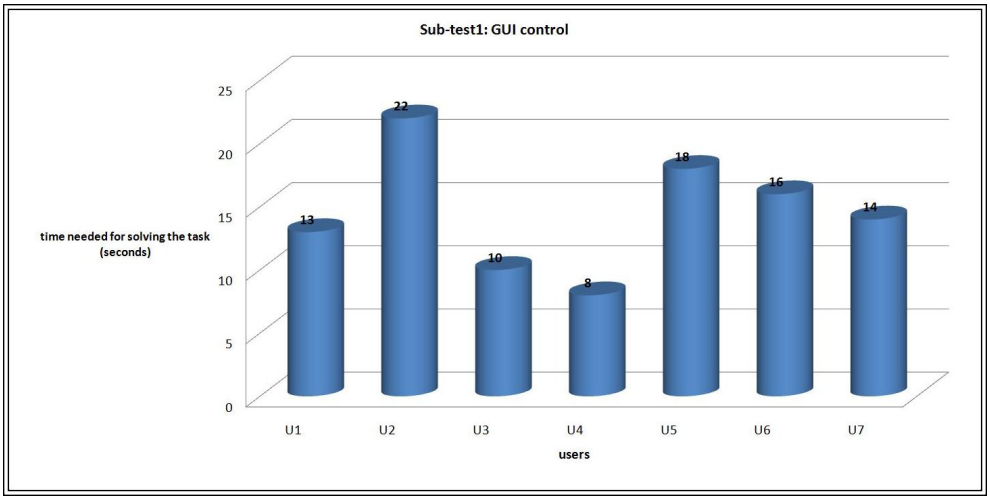


Figure 6.2: Sub-test1 - diagram

Sub-test2 scenario: time for learning the algorithm

The goal of this test, was to give a user time to learn the principles of Minimax and Alpha-beta pruning algorithms. After the user said, that he understood these principles, he was subsequently asked to solve two following independent tasks.

In the diagram 6.3 you can see the times that the individual users needed for understanding the game algorithm principles. The columns marked as **Min** mean the time taken for learning the Minimax algorithm and **Alph** for Alpha-beta pruning.

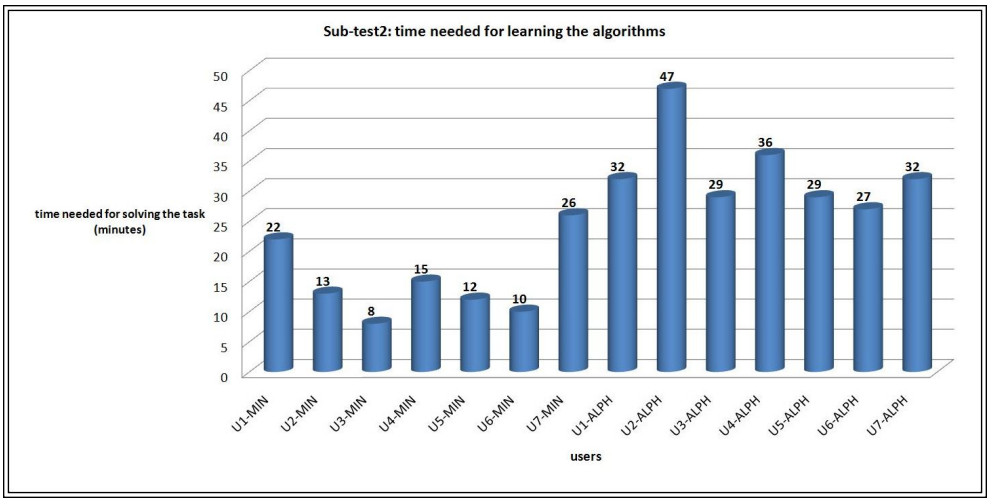


Figure 6.3: Sub-test2 - diagram

Sub-test3 scenario: solving the task without using of the application

In this sub-test, the users were asked to solve the simple game situation (for all users the task was the same) by the Minimax and Alpha-beta pruning algorithms, on the paper, without using the implemented application. The goal of this test was to find out, how

many errors the users will make, and compare this result with the situation, when the users use the application.

In the diagram 6.4 you can see the count of errors the users made while solving this task. The colums marked as **Min** mean the count of errors while solving the task by the Minimax method and **Alph** by Alpha-beta pruning.

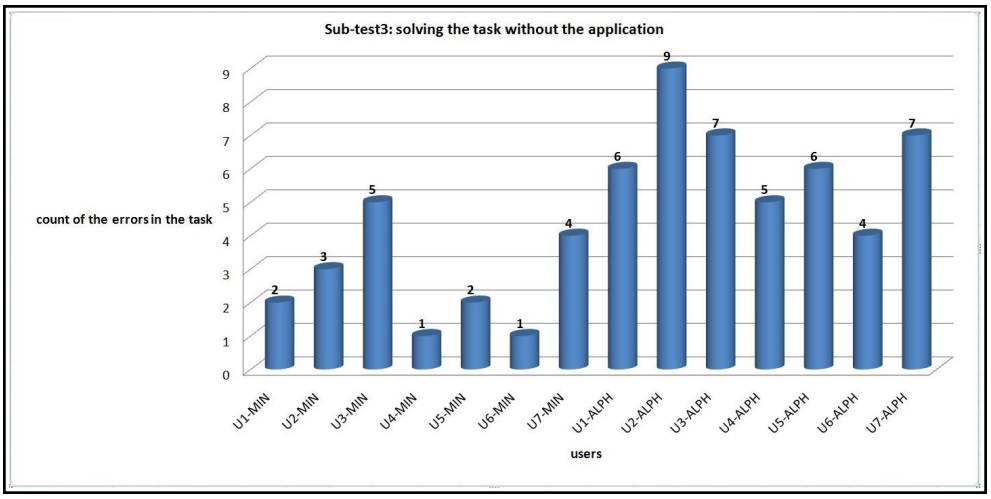


Figure 6.4: Sub-test3 - diagram

Sub-test4 scenario: solving the task with using of the application

In this sub-test the users were also asked to solve the different, simple game situation (for all users the task was the same) on the paper, but they used the implemented application while solving this task. The users could not solve the whole task by the application, they could only review the status of their solution.

In the diagram 6.5 you can see the count of the errors the users made while solving of this task.

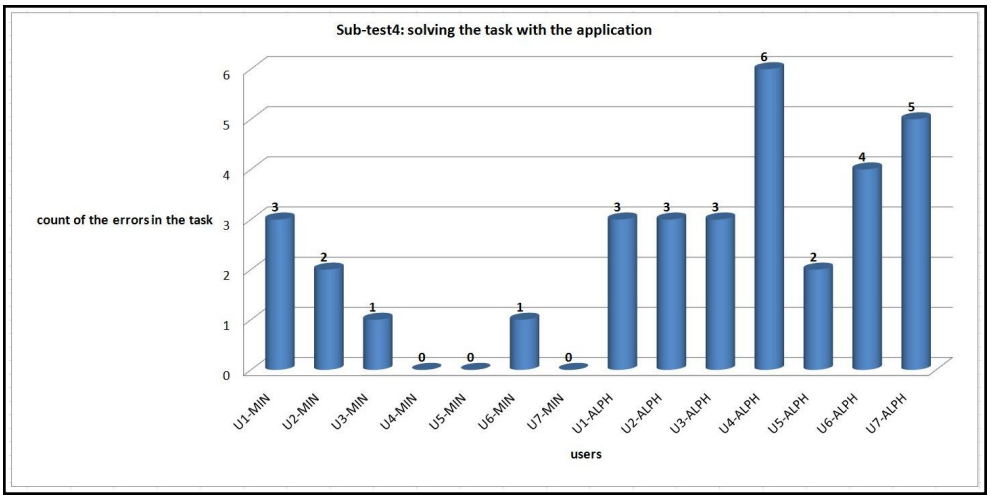


Figure 6.5: Sub-test4 - diagram

6.3 Evaluation of the tests results

During each test, I was physically present and I watched the user's behavior during his application control activity. I focused on these two following aspects:

- **time needed for understanding the application control** - each user, after he saw the application for the first time, did not have a clue how the application works and needed to read the application help. After reading the help, each user was able to control the application more or less immediately. Based on this observation flows, that the average time needed for understanding the application control system is 14,42 seconds.

- **time needed for understanding and the level of understanding of the game algorithms** - it turned out, that for the users who did not have any clue about the artificial intelligence, the graphical demonstration of the algorithm principle, was not enough. For that reason it turned out as a good idea, to implement the window, where the user can read the basic information about the chosen game algorithm in pdf form. Each user took this option, and after reading this information, in combination with the graphical demonstration, understood how the implemented game algorithms work. At first, the application seemed a bit confusing for the users, but after some time it started to make sense for them. Based on this observation flows, that the average time needed for understanding the Minimax algorithm principle is 15,14 minutes and time needed for understanding the Alpha-beta pruning algorithm principle is 36,85 minutes. If we take into account the number of errors the users made, while solving the test tasks, when the users could not use the application, the average count of the errors was 2.57 per user in the Minimax and 6,28 in the Alpha-beta pruning. On the other hand, when the users could use the application, and could review the status of their solution, the average count of the errors was 1 per user in the Minimax and 3,74 in the Alpha-beta pruning.

Of course the testing on sample of 7 users can not provide completely relevant results. For that reason it would be beneficial to distribute the application for the greater mass of the users.

Since the application is focused on narrow group of the users, predominantly the students of the artificial intelligence, it would be suitable to distribute the application for example on the web pages of these courses etc. Nevertheless, it turned out, that on the base of this application, it is possible to better understand the principles of Minimax and Alpha-beta pruning algorithms, in rather short time interval.

All of these users, on which the application was tested, agreed, that this application represents the good learning "complement", which could save their time, during the learning of the game algorithm principles.

Chapter 7

Conclusion

This chapter discusses the achieved test result mentioned in the chapter 6.3 and subsequently in this chapter are mentioned the proposed future improvements of the application.

7.1 Proposal for the possible future improvements

Even thought the developed application is really usable and gives a fairly satisfactory results, there are still many things that could be improved.

In terms of improving the application at the source code level, it would be worth to consider the optimization of algorithms, for generating the graphical output. For example, in case of the tree consisting of the units up to tens of nodes, the application response time is almost instant. In case of the tree consisting of the hundreds up to thousands nodes, the application response time reaches tens of seconds.

Regarding the new functions of the application, it would be good idea to implement the additional concept describing the principle of the methods for solving the tasks for one player, for example the Tower of Hanoi game 3.1.

7.2 Discussion of the achieved results

It turned out, that for the users who do not have any clue about the artificial intelligence, the principle of the graphical demonstration of the game algorithm, is not enough and this application will hardly replace exclusively study materials or even the teachers.

On the other hand, this application is the good learning „complement“, which allows the users to make sure, that they really understand the game algorithm principles and it also allows to deepen their knowledge.

Since as the goal of this thesis was to develop the application for educational **support**, the discovering that the developed software can not fully replace exclusively study materials, is not unexpected.

Bibliography

[1] Prohledávání stavového prostoru [online]. http://cs.wikipedia.org/wiki/Prohledávání_stavového_prostoru, 2013-10-15 [cit. 2015-05-27].

[2] Minimax (algoritmus) [online]. https://cs.wikipedia.org/wiki/Minimax_(algoritmus), 2015-05-02 [cit. 2015-05-27].

[3] Software framework [online]. https://en.wikipedia.org/wiki/Software_framework, 2015-05-20 [cit. 2015-05-27].

[4] C++ [online]. https://en.wikipedia.org/wiki/C%2B%2B, 2015-06-11 [cit. 2015-05-27].

[5] Stavový prostor a jeho prohledávání [online]. https://cw.fel.cvut.cz/wiki/_media/courses/y33zui/01_neinformprohled_v2.pdf, [cit. 2015-05-26].

[6] Traced by User:Stannered. A sample tic-tac-toe game, for en. [online]. http://en.wikipedia.org/wiki/File:Tic-tac-toe-game-1.svg, 2007-03-30 [cit. 2015-05-27].

[7] Kristian kraljic. Game visualization [online]. http://kra.lc/projects/gamevisual/launch.php, 2011 [cit. 2015-05-27].

[8] Václav Matoušek. Hraní her (teorie a algoritmy hraní her) [online]. http://www.kiv.zcu.cz/studies/predmety/uir/predn/P2/FThema2_hry.pdf, 2015-03-04 [cit. 2015-05-21].

[9] Jan Němec. Složitost alfabeta metody [online]. http://www.linuxsoft.cz/article.php?id_article=1239, 2006-07-17 [cit. 2015-05-27].

[10] Ondřej Popelka. α -β prořezávání [online]. https://akela.mendelu.cz/~xpopelka/cs/ui/prorezavani/, 2015-01-08 [cit. 2015-05-27].

[11] Tomáš Ripel. Řešení úloh rozkladem na podproblémy [online]. http://1url.cz/V5o0, 2009-05-08 [cit. 2015-05-27].

[12] José Manuel Torres. Demo: minimax game search algorithm with alpha-beta pruning (using html5, canvas, javascript, css) [online]. http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html, 2011 [cit. 2015-05-27].

[13] Vladimír Mařík. Olga Štěpánková, Jiří Lažanský a kolektiv. *Umělá inteligence (1)*. Academia, 2000. ISBN 80-200-0496-3.

[14] F. Zbořil and F. Zbořil ml. Základy umělé inteligence izu studijní opora [online]. https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-5a.pdf, 2013-04-30 [cit. 2015-05-21].

Appendix A

CD content

CD directory structure

- **SRC** - directory containing the source codes of the application.
- **Thesis** - directory containing the LaTeXsource codes.
- **Video** - directory containing the short video presentation.
- **README.txt** - text file containing the information about compiling the application.
- **bachelor_thesis_xcasla03.pdf** - text of the thesis.

Appendix B

Manual

Process of starting the application

- **Compilation** - For compiling the application you need installed *QT* framework version 3.2.0 or higher and compiler of programming language *C++* version 10 or higher. Application is possible to compile on platforms *Microsoft Windows* or *Linux*.
- **Application control** - After starting the application, user must prepare his initial game situation. By clicking the right mouse button on the game board squares, he can place the game marks onto the game board. The only condition is, the last placed game mark must be opponent's mark (circle). After preparing this initial situation the user can choose one of two implemented game algorithms (Minimax and alpha-beta pruning) from the menu and start generation of the application output. If the application output is generated, the user can modify it by clicking the **next step** button and simulate the game algorithm activity. If the user wants to see, which game move the game algorithm chose as the best one, he can click on the **place opponent move** button, and the opponent's mark will be shown in the game board on its best position. More detailed information about the application control system you can read in the chapter 4.4.

Appendix C

Application screenshots

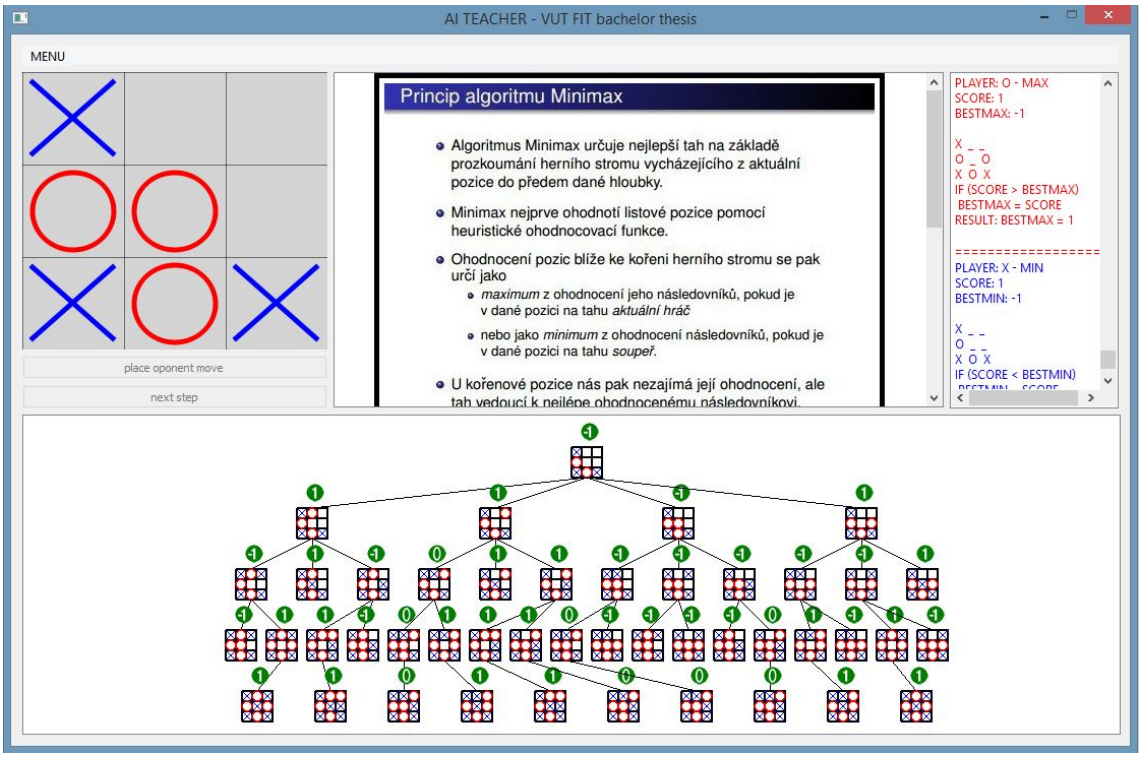


Figure C.1: Application screenshot - Minimax algorithm

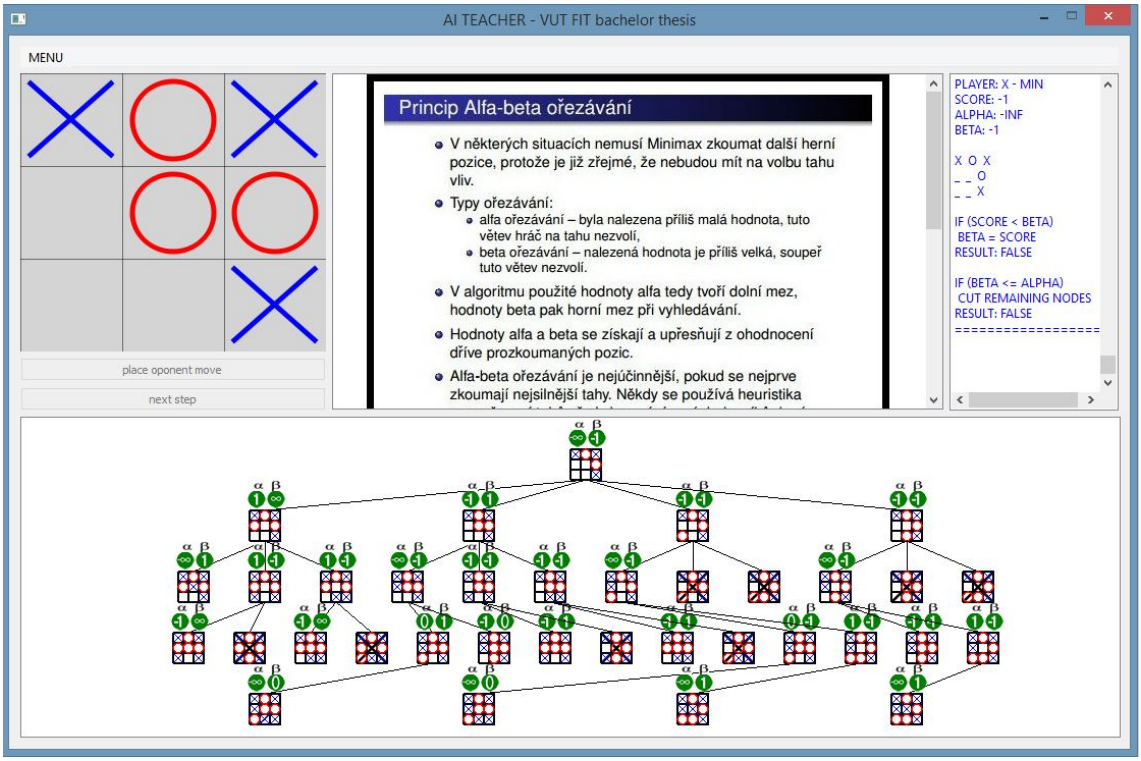


Figure C.2: Application screenshot - Alpha-beta pruning algorithm