*Alexander Simeonov*

# Learning by Playing – – Creating Custom Board Games Framework

*The purpose of this project is to create a basic framework that will allow players to create their own board games by designing the boards and the pieces, defining the rules and the winning conditions. Further on, the games will be tested against other human players, and when suitable winning strategies are discovered and formalized, the users will be able to design their own computer opponents. So far, a simple two-player game of checkers has been implemented.*

## 1. Introduction

### 1.1. Brief history of board games

Games have been a very important part of human life since ancient times. Many games have been played by using various kinds of boards and pieces — archaeologists have found boards and pieces that were used thousands of years ago.

### 1.2. Games as an educational tool

Games are not played only for entertainment — they have proven to be very useful as educational tools [2]. Playing board games requires different mental skills. Chess, for example, is believed to be a great game for teaching children to think strategically, i.e. to try to predict the moves their opponent will take and make moves preventing the loss of men.

## 2. Project goals

With the ever increasing speed of computers, many "artificially intelligent" applications are being created to play different types of strategy games against humans [5]. Creating a suitable environment in which the player could model various sets of rules for board games and studying the corresponding winning strategies has great educational potential.

*Alexander Simeonov, Bulgaria, 9004 Varna, Neofit Rilski Str., Nr. 15, ap. 8, Mathematics High School "D-r Petar Beron", Varna*

The purpose of this project is to create a basic framework that will allow players to create their own board games by designing the boards and the pieces, defining the rules and the winning conditions. Then, the games will be tested against other human players, and when suitable winning strategies are discovered and formalized, the users will be able to design their own computer opponents. The framework will include some examples of board games so that players could study the main board game design principles, before they start creating their own games.

## 2.1. First step: creating the board

Creating a board designing feature in the framework is an essential part of the project. The basic functionality needed by a board designer should provide the following facilities:

- Adjusting the board size
- Coloring the board cells
- Putting various obstacles on the board (e.g. walls between or inside cells)
- Change cell shape (rectangular, triangular, hexagonal)

Except for defining boards for custom games, it may be very interesting for the users to study how a particular game's rules should be modified so that the game can be played on a different board — e.g., a hexagonal chess board, or extended 16x16 checkers board could make the game a lot more challenging for both players.

## 2.2. Defining the pieces

The pieces and the moves they are allowed to perform are probably the most important part of each game. The pieces editor allows users to define the properties of the pieces they need to play their game. This includes shape or picture, directions in which they move and directions in which they capture other pieces.

## 2.3. Testing the created game

Trying to create computer opponents for a game that has not been played before is a very difficult task. Therefore, users are expected to test the game with human opponents and find out if a particular strategy is good before trying to formalize it for the computer opponent.

So, after the game rules are defined, the game needs to be tested in the framework's Test Mode. This is a special mode that has some special features allowing fine tuning the game in real time, like the ability to move pieces around the board without obeying the rules, or adding new pieces, so that the game can be started from a particular state (piece configuration on the board).

## 2.4. Creating the opponent

Many algorithms for implementing computer opponents in various strategy games have been developed during the past [3], but it was not until 1997 that the world chess champion lost against a computer [5]. This is because computers, being much faster than a human, try to examine every possible move; however, a few decades ago it was not possible to create machines that were good enough to beat humans because they did not have enough computing power. In the following few sections, some clarifications about the AI principles used in the framework will be given.

### 2.4.1. Game trees

One of the fundamental game theory theorems is the Minimax theorem, stating that for games satisfying some technical conditions, there always exists an optimal strategy. For more details, see [4]. Such a strategy can be discovered using game trees.

Knowing the current piece positions on the board and the rules of the game (how pieces move and capture), the playing program, by examining all possible moves and counter moves, generates a tree in which each node represents the game state after the particular move was played. If the game tree is complete (i.e. all the tree leaves represent a game terminal state that is either a victory for one the players or a draw) we can always find an optimal strategy by using the minimax algorithm.

### 2.4.2. Searching the game tree

When the whole tree is generated, and all leaves are assigned a specific value (i.e. “win”, “draw” or “loss”), the minimax algorithm has to be applied. We will assign the following number values to the final game states:

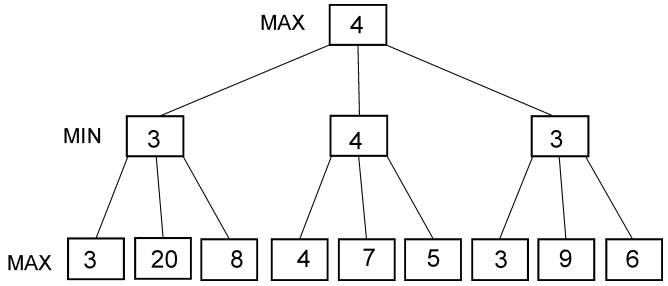
| | |
|---|---|
| Win | 1 |
| Draw | 0 |
| Loss | –1 |

The algorithm assumes that during our turn, we try to maximize our gain (i.e. to go to a node of a higher value), while our opponent will try to minimize it during his turn. Thus, all the nodes on a tree-level representing the opponent’s turn are called MIN nodes, and all the nodes on a tree-level representing our turn are called MAX nodes. Starting from the tree leaves, we go one level up during each turn and assign values to the nodes of the current level by the following rules:

– If it is a MIN level, assign the minimum value of all ancestors’ values.

– If it is a MAX level, assign the maximum value of all ancestors' values.

The best move (assuming both players always choos the best move available) ends up at the root of the tree (see figure).




Minimax game tree

## 2.4.3. The evaluation function

Generating the complete game tree and going through all the moves, however, is not an option for most strategy games, because of the relatively large tree branching factor.

For some games (e.g. Nim), there are known algorithms that allow checking if the current game state is a winning one without going down the tree branch until a leaf is reached. So, we can only generate a tree one level deep and still be able to choose the right branch. However, no such algorithm is known for most of the more complex games (e.g. chess). Therefore, we need a heuristic approach; we could use a function to calculate the probability for a game state being winning. This will be our evaluation function.

Such evaluation function for a chess game may look like this:

$$C_1 * \text{KingSafety}() + C_2 * \text{PiecesMobility}() + \\ +C_3 * \text{CenterControl}() + C_4 * \text{PiecesValue}() + \ldots$$

where $C_i$ are some weight-coefficients provided by the user to specify the importance of the particular game factors [6]. The higher the resulting value, the better the player's position.

Using this function, the computer can look up a given depth further in the tree and decide which moves will ensure its dominance when the level is reached. However, a possible domination may not be final. A few levels further into the tree and what looks like a winning position may turn out to be a losing one. The conclusion is that the deeper we go into the tree the better the computer's moves will be.

Sometimes not only the examined depth matters. If the evaluation function is not good enough, and the results are wrong, a more shallow

search with a better evaluation function will give better results. One big benefit of the feature allowing users to define their evaluation functions is that they could study and explore the strategies' effectiveness and the game factors' importance by creating different computer opponents and running them to play against each other.

### 2.4.4. Optimization

When the game branching factor is rather high, (as e.g. in the Japanese GO game in which we have about 360 branches per node [3]) even a few-level depth search might make the game tree grow far too fast. There are some algorithms that allow decreasing the number of branches whose leaves need to be evaluated. The alpha-beta pruning [7] algorithm is an example of algorithm that "cuts off" the branches that seem unlikely to give good results. Reducing the number of leaves to be evaluated allows going deeper in the tree without using more processor power. Usually the branching factor is decreased to its square root, which allows doubling depth level. The specifics of the algorithm will not be discussed here, but can be found in [7].

## 3. Current implementation

Because of the need for a modern and platform-independent object-oriented language during the developing process, Java is being used. Thus, the game framework may be placed as an applet on a web page and the only requirements for users to run it will be a web browser and Internet access. Furthermore, there are many useful APIs implemented for Java (e.g. the QuickTime video API), which will speed up the development process.

Together with developing good enough computer opponents and flexible tools for defining board games in the framework, we put a lot of effort into creating a user-friendly interface by exploring the potential of the available hardware.

### 3.1. User input interface

Board games were first played with small tangible objects on real boards. In order to combine the advantages of physical manipulation of objects [10] with the power of computer analysis of winning strategies, we use the Sense Board, an 11.5 m board with a grid (1220 cells), connected to a computer running the game application via serial cable. Magnetic tokens are attached to the board and used as pieces in the game. Every piece move is detected by the board and sent to the computer.

Players can set additional parameters to their pieces by dialing a three-digit number on some of the tokens during the game is played (e.g. normal or king status in a game of checkers).

The board will read the radio frequency identification tags (RFIDs) of the small magnetic pieces. The communication protocol is a fairly simple poll-based one — the program asks the sense board if the board has received any signals, and if this is the case, the changes are reported.

## 3.2. Projection on the board

A possible approach to display the user-designed game boards on the Sense Board would be to print large sheets of paper and stick them to the board. However, this would require lots of printing supplies, and would make the board hard to modify in real time — if the board-structure is affected in some way by a piece (e.g. an obstacle is removed or added), the parts of the cover will have to be reprinted and put as patches on the old board.

Another good solution exists — using a multimedia projector to display on the board will allow whatever is shown on the computer screen to be drawn on the board. There are virtually no limitations with graphics and animations that need to be displayed.

The projector approach was implemented because it worked better. To make the game even closer to real board games, the sense board was detached from its frame and put on a table. Having in mind that most of the projectors are designed to project on vertical surfaces (e.g. walls), we had to find a way to project on a horizontal flat surface. To avoid attaching the multimedia projector to the ceiling, we built a tall metal construction with a mirror attached on its top, thus allowing projection from the ground to be reflected on the board positioned next to the projector. Some difficulties occurred while testing this method — because the mirror was not parallel to the board, the projected image had the shape of a trapezoid and the board cells did not fit. This problem was solved by using the projector's keystone feature, which transformed the projected picture into a rectangular shape again.

## 3.3. Game replays analysis

Replaying games or parts of games is often a very useful facility. Apart from entertainment reasons, they are an essential part of improving player's strategy. Lots of people download popular computer games' (such as StarCraft or WarCraft) replays, recorded from tournaments and watch them to see what exactly led to victory, so that they could apply the same "tricks" later.

Watching one's own games is also a learning opportunity. Furthermore, the computer analysis of the game may be used to highlight only

interesting moments from the video and thus make a long game shorter. In a chess game notations, the particularly good moves are marked with an "!", and moves considered bad are marked with a "?". A similar notation system will be introduced in the framework. To make this possible, interesting moves have to be differentiated from the regular ones. The game tree and the evaluation function can be used — if a player made the best move according to the computer-generated minimax tree, it can be considered interesting. Or, on the other hand, taking poor branches that lead to defeat could be highlighted, too.

### 3.4. Video capturing system

Implementing a video capturing system seemed a good idea and turned out to be easy to implement. Three web cameras attached to the computer's USB ports are used to capture video of the game from different angles. Because of the limitations of the Java virtual machines (only one web camera can be used by a single VM), different processes have to be created if the user wants to capture with all the cameras simultaneously.

The cameras capture the whole game first, and after it is over, the videos are automatically split into smaller files containing single moves.

## 4. Conclusion

Because of the large complexity of the framework, we were not able to finish all parts of the project. So far, just a simple game of checkers has been implemented as a test game. The Sense Board and the projector interface worked reasonably well and proved to be very user-friendly. The AI system is in its very early stage and is not optimized enough to be usable in real games. The video cameras, in harmony with predictions, captured both players' emotional states during the game.

The game board framework is in its early stage of development and has lots of features that need implementation or improvement (e.g. equip players with monitors that display the minimax game trees during the game play); they will be available in the following versions of the application.

**Acknowledgments**

First, I would like to thank my mentor, Mr. Walter Bender, for assigning me to work on this project, for helping me, giving directions on my paper and supplying me with all the hardware I needed.

I want to thank Erik Blankinship, the graduate student I was working with during almost all of the time. His support, ideas, and useful advice were really invaluable.

Many thanks to my tutor, Chris Mihelich, for the corrections on all my paper drafts and presentation speeches.

Also, I want to thank the "St. Cyril and Methodius" foundation for the financial support that made my participation in RSI possible.

My very deep gratitude to Matt Paschke, Dr. Jenny Sendova and all other RSI and CEE staff people for the assistance, guidance and the support they gave me during the whole program.

Thanks go also to all the counselors (especially Logan Dean) for helping me to forget for a while about the research and relax when I needed to.

To the "nobodies" Andy and Linda a sincere "Thank you" for reading my paper and giving helpful feedback.

---

## References

[1] 2004. The royal game of Ur and Tao. Available at http://www.tradgames.org.uk/games/Royal-Game-Ur.htm

[2] 1985. Learning and mathematics games. Journal for research in mathematics education, Monograph, **1**, NCTM, Reston, VA

[3] Andr Schoorl. 2004. Game playing. Available at http://engr.uvic.ca/časchoorl/~eng420/notes/Game_Playing.pdf

[4] Eric W. Weisstein. 2004. Minimax theorem. Available at http://mathworld.wolfram.com/MinimaxTheorem.html

[5] IBM Research Team. 2004. Deep blue. Available at http://research.ibm.com/deepblue/

[6] Wikipedia. 2004. Evaluation function. Available at http://en.wikipedia.org/wiki/Evaluation_function

[7] Wikipedia. 2004. Alpha-beta pruning. Available at http://en.wikipedia.org/wiki/Alpha-beta_pruning

[8] Henry Bottomley. 2004. How many tic-tac-toe (noughts and crosses) games? Available at http://www.research.ibm.com/~sw16/hgb/tictac-toe.htm

[9] Eric W. Weisstein. 2004. Chess. Available at http://mathworld.wolfram.com/Chess.html

[10] Robert J.K. Jacob, Hiroshi Ishii, Gian Pangaro, and James Patten. 2001. A tangible interface for organizing information using a grid. Proceedings of CHI, ACM Press