Developing a New Java Algorithm for Playing Backgammon

Manuela Panoiu, Caius Panoiu, Ionel Muscalagiu, Anca Iordan and Raluca Rob

***Abstract*—** A computer game is a very convenient way of recreation. In order to simulate most classical games, many algorithms have been implemented. The complexity of algorithms used in implementing the games leads to a continuous increasing of the computer performance. The application presented in this paper is able to play backgammon. The software allows a game between two players and also a game between one player and the computer. A software package module allows monitoring games in the network. All software programs were implemented in Java language.

***Keywords*—** Heuristic algorithms, computer games, backgammon, java..

I. Introduction

COMPUTER games have concerned many software developers and also many researchers in artificial intelligence. Games are also an important application of heuristic algorithms. Backgammon is a game with the same chance as Bridge, Scrabble, Poker. Many algorithms for playing such games have been implemented. Because working with incomplete and imperfect information such an algorithm is difficult to implement.

The early computer programs for backgammon were knowledge based. [1], [2]. In these systems searching techniques are not very much used. This happens because backgammon has a large state space (more than 10^{20} states [1]) and a branching factor imposed by 21 dice rolls. The first computer program for backgammon was BKG 9.8 [6]. It was programmed by Hans Berliner in 1970 on a PDP-10. Early versions of BKG played badly even against beginner players. [6]. There are some typical algorithms used for implement computer programs for playing backgammon. Such an algorithm is minmax algorithm [7]. Minimax is a generalization of Alpha-Beta search for minimax trees with chance nodes. [7]. In case of games with chance a variation of minmax algorithm can be used. Such an algorithm is expectiminmax [8]. Other computer programs for backgammon have used artificial neural networks [3], [4] and [5]. Such an example is Neurogammon [2] that was trained with supervised training. An improved version of this program was TD-Gammon [1] based also on artificial neural networks.

Our computer program is based on searching techniques, a variant on min-max algorithm. The program has two components: a component for playing backgammon and a component for monitoring games in the network. In the next section the software will be described.

II. Backgammon – A Game of Chance

Backgammon is a nondeterministic game with chance. In nondeterministic games, chance is introduced by dice or card-shuffling. In case of Backgammon the chance is introduces by dice. Playing backgammon is simple: there are two players with 15 pieces each. The final goal is to move all pieces off the board. The rules are: Dice roll determines number of moves, players move in opposite directions, pieces cannot put on a point occupied by 2 or more of opponent's pieces, single piece can be "hit" if it is put on the same line a piece of the opponent and hit piece must start a new route. The backgammon board is shown in figure 1.

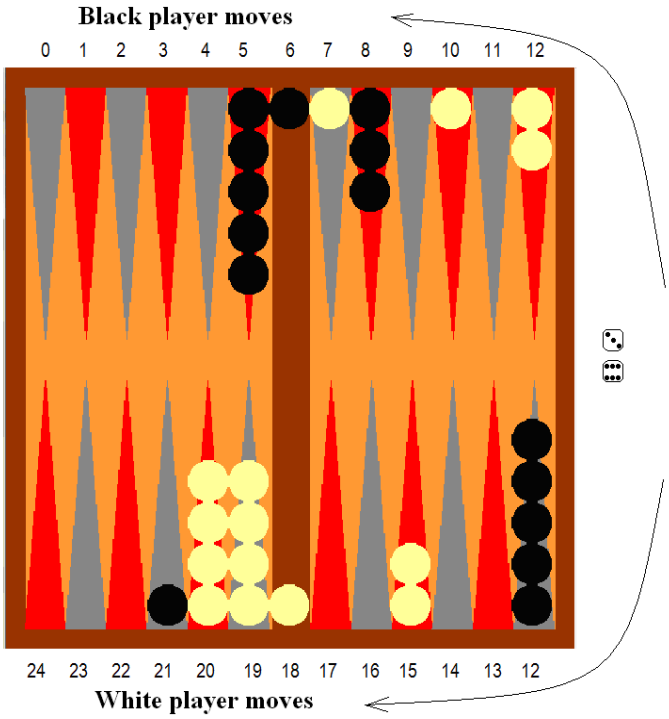

Fig. 1 The backgammon board

There are some typical algorithms used for implement computer programs for playing backgammon. Such an algorithm is min max algorithm [7]. Minimax is a generalization of Alpha-Beta search for minimax trees with chance nodes. [7]. In case of games with chance a variation of min-max algorithm can be used. Such an algorithm is expectiminmax. In this algorithm, the outcome depends of a

combination of the player's skill and chance elements such as dice [8]. The expectiminimax tree is a specialized variation of a minimax game tree that plays two-player zero-sum games (such as backgammon). In addition to "min" and "max" nodes of the traditional minimax tree, this variant has "chance" nodes, which take the expected value of a random event occurring [8], [9]. The form of the game tree for backgammon is shown in fig. 2 [9].

A. Object oriented modeling of the software

The software was design using OOP modeling. Using UML it was design the use case diagram shown in figure 3.

Object oriented design uses the representation of the static structure of the backgammon software using classes and the relationship between them [11].

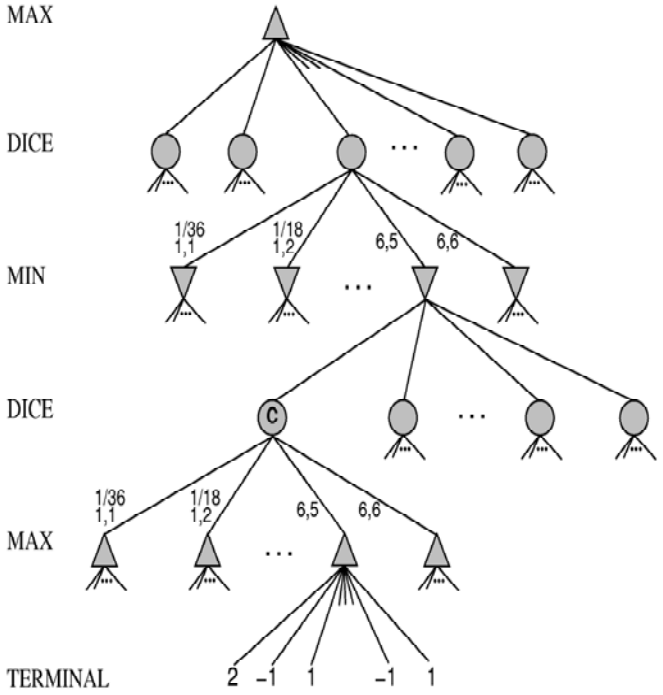

Fig. 2. Schematic game tree for backgammon position

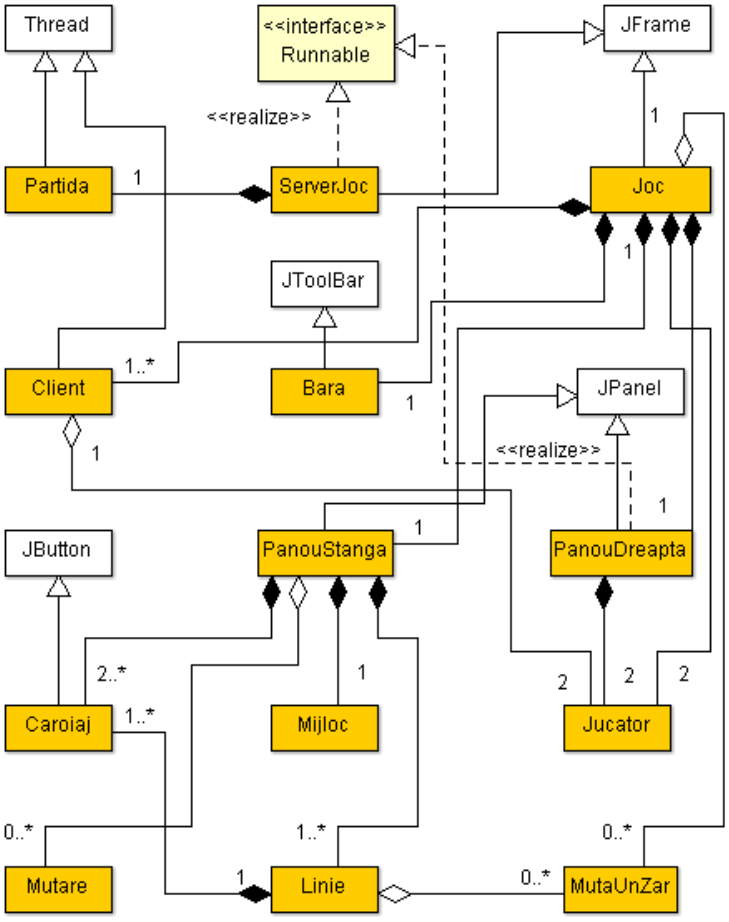

Fig. 3. Use case diagram

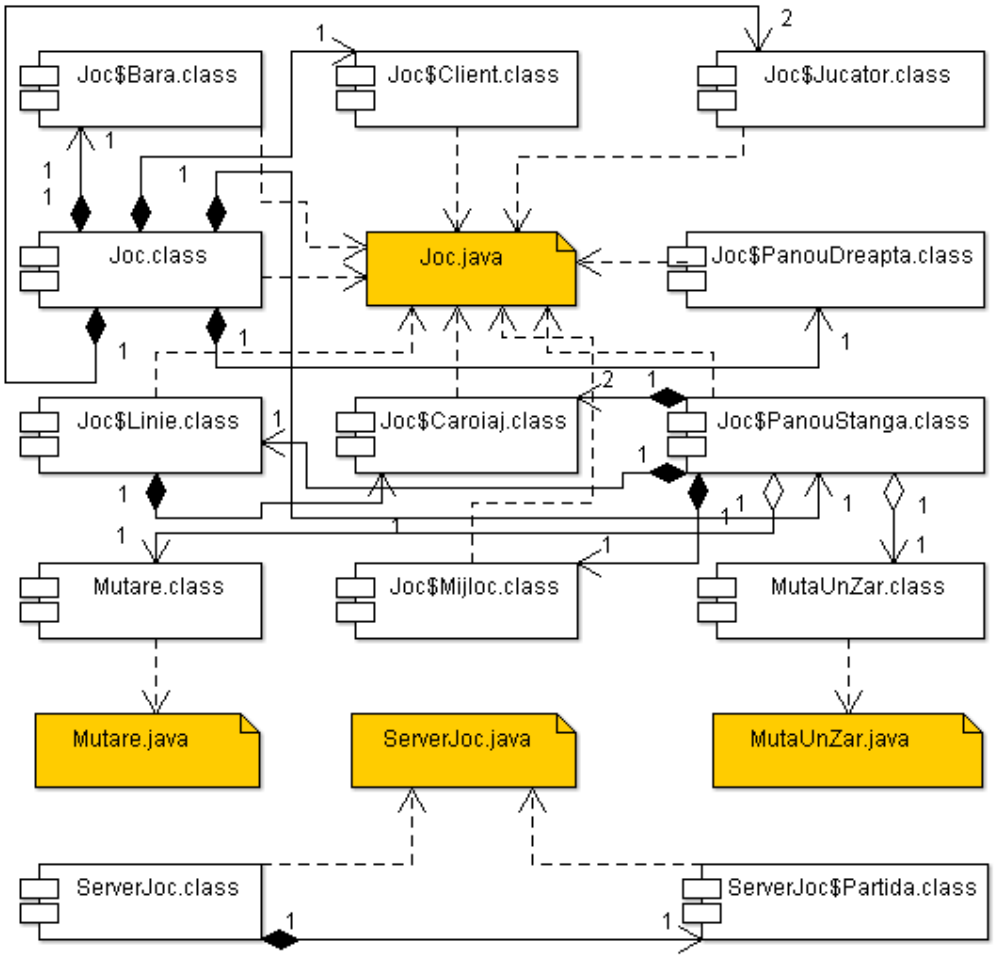

Fig. 4. The component diagram

B. Algorithm for Backgammon playing

Backgammon is a game with chance introduced by dice. The board implemented using java swing classes is show in figure 5. As can be seen, the board is very similar with a real backgammon board. In the right panel dice throwing was simulating. For doing this probability theory, it was used a method described in [11] and [12]. So, for each dice was considered the follow:

$$\text{int}(rand(\,)*6)+1+\text{int}(rand(\,)+6)+1\,. \qquad (1)$$

Like is shown in [12], this is the correct way to do the dice throw simulation instead generating two individual random numbers.

The algorithm uses three difficulty levels:

- "beginner player"
- "intermediate player"
- "experimented player"

These levels are used for simulate the ability of the second player (computer algorithm).

For the "beginner player" level we don't use any intelligent techniques. This level was designed in order to be useful for those who want to learn the rules of backgammon. The algorithm simply picks a random move from the list of all legal movements and performs it. This option of the software can be used for the beginner players to learn to play backgammon.

The "Intermediate player" level uses a heuristic algorithm to select the next move for the computer. This algorithm uses a list of allowed movements of the current state of the game. For all these movements in the list there is assigned an evaluation function that calculates a numerical value called "merit factor". This evaluation function depends on several conditions such as: the number of pieces of the player that remain unprotected, stay in pairs, the possibilities to build a gate. The algorithm is given below.

The method DetMeritFactor returns the merit factor for a state S_1. This method is a variant of the min-max algorithm, an algorithm frequently used in implementing games. The algorithm is shown in the follow.

```
Playing Algorithm
Initialize the board with all 15 white
pieces and 15 black pieces
  While no player has wont yet
    Next_moves←{}
    for each possible move from the
    current state S,
      determines the next state, S1,
      of the board
       next_moves←next_moves ∪S1
    for each state from the next_moves
    list do
      merit_factor←DetMeritFactor(S1)
    MaximMerit←maxim(Merit)
    Perform the move for the state with
    MaximMerit merit_factor and obtain
    NewPosition
    If NewPosition is winning state
        Break
    Read the opposite player's move
    and  modify the board
  end
```

```
DetMeritFactor(S1)
  If the state S1 is a winning state
    Return the maximum default value
for merit_factor
  For each posible moves of the
opposite player from the
  state S1
    Calculate the merit factor Mi for
each state resulted
    from these moves
  Select the state with the minimum
merit factor Mmin,
  considering that the opposite will
made the most defavorable move
  Return Mmin
End
```

For this level it wasn't assigned a better evaluation function value for the movements that can remove opposite player pieces from the board. These movements are allowed like any other movement. The algorithm inspects several moves in advance to determine the sequence of movements that will lead to win the game.

The "Expert player" level is not very much different from the second level. The main difference is that the algorithm for "computer" in some cases takes the chance and hit a piece of the opponent even if some of his own pieces remain unprotected. Moreover, such a movement is preferred by the "computer" player (algorithm).

C. The software package

In the left panel is shown the board for playing backgammon; the board is shown like at the beginning of game.

The software package was implemented in Java programming language, because of the advantages of this language: is simple, platform – independent, distributed, secured and robust. Several classes have been implemented for modeling the players, the table game, the game, the moves, a ToolBar class, a class for the left and right panel, etc.

The graphical user interface, shown in figure 5, has a menu bar with the following options: File, Game, Options and Help. The "File" option allows through a drop down menu to start a new game, save a game or load a saved game and to exit the application. In case of saving a game, actually the current state of the game is saved (the position of all pieces, the temporary remove pieces and the next player that will throw the dice). This information can be anytime recovered when a game is load. In this case a game can be continued just from the stage when it was interrupted.

In order to start a game, first of all, the user must select the players from the "Game" menu. This menu has two options: one player or two players. In case of selecting two players, these are human users that will play the game according with their abilities of playing backgammon. In case of selecting one player, the first player is the human user and the second player is a heuristic algorithm that will be described in the following section.

In order to represent correctly the position of pieces on the board, the lines were numbered from 1 to 24. So, for a single piece, their position is stored through the piece color and line number. The number of pieces from a line is also stored using `"Linie"` class.

In figure 6 is shown how the position of a piece is stored. Line 6 is used for temporary removing pieces from the board. For all legal moves storing a class `"MoveDices"` was designed. Using this class was describing a movement. The class contains a variable array to store dice values and the position for the moved piece. This variable array has four lines and three columns. The first two lines store the dice values. The last two lines are used only in case of double dice movement. For storing a movement of one piece it was used: the value of the dice, the line of start and the color of the piece.

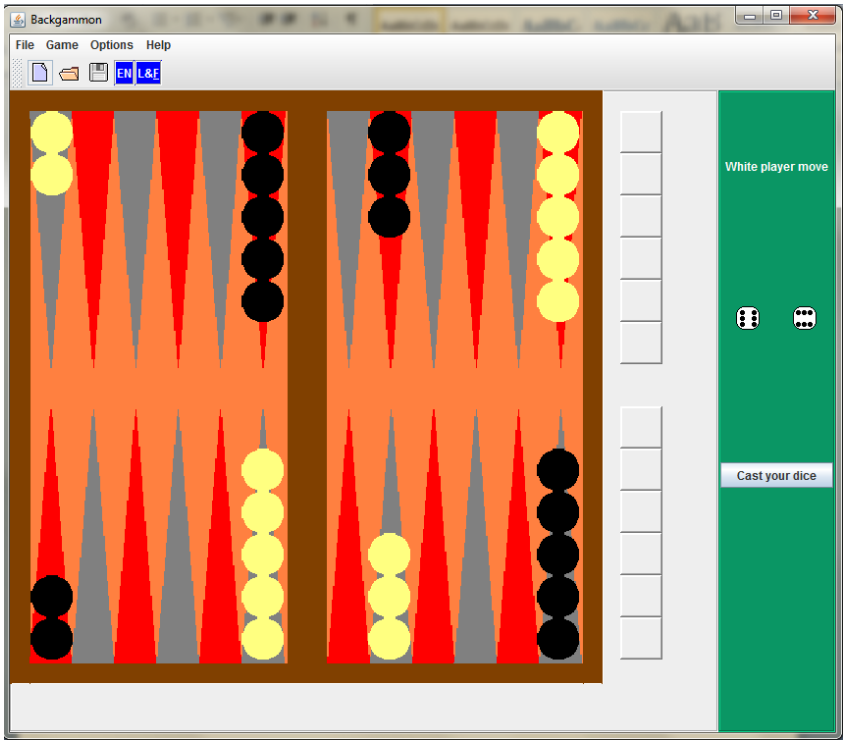

Fig. 5. The Backgammon board for playing

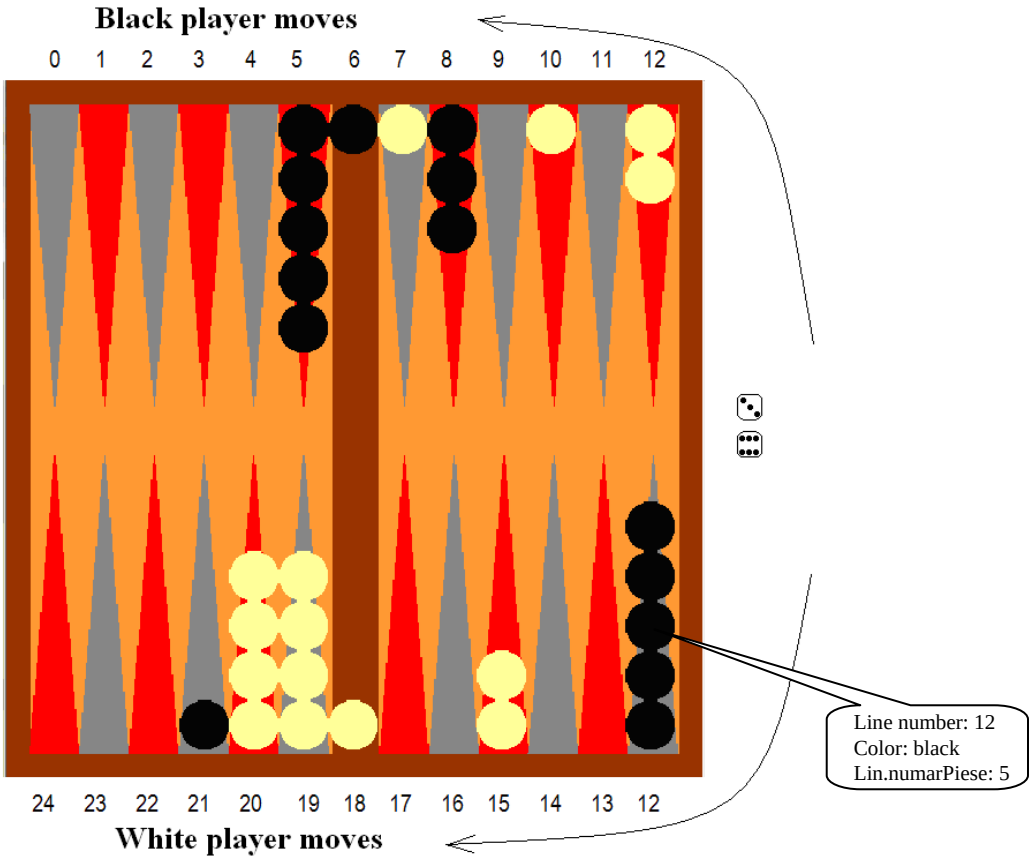

Fig. 6. Representing pieces positions

There are configurations of the board in association with certain dice values when it is possible to move a single piece. For these cases class named "`MoveADice`" was also designed and implements. The class is a simplified version of the previous class MoveDice.

In figure 7 is shown a configuration of a game in case of using "Intermediate player" level. In the figure 6 is shown the next configuration. In both, figure 7 and 8, the computer program software uses black pieces and human users use white pieces. So, in figure 7 is shown the configuration of a game in case of 5 – 2 dice values for the white pieces (human player). It can be observed that the player was taken out an opponent piece from the board. In figure 8 is shown the next movement selected by the algorithm in case of 4 – 1 dice values.

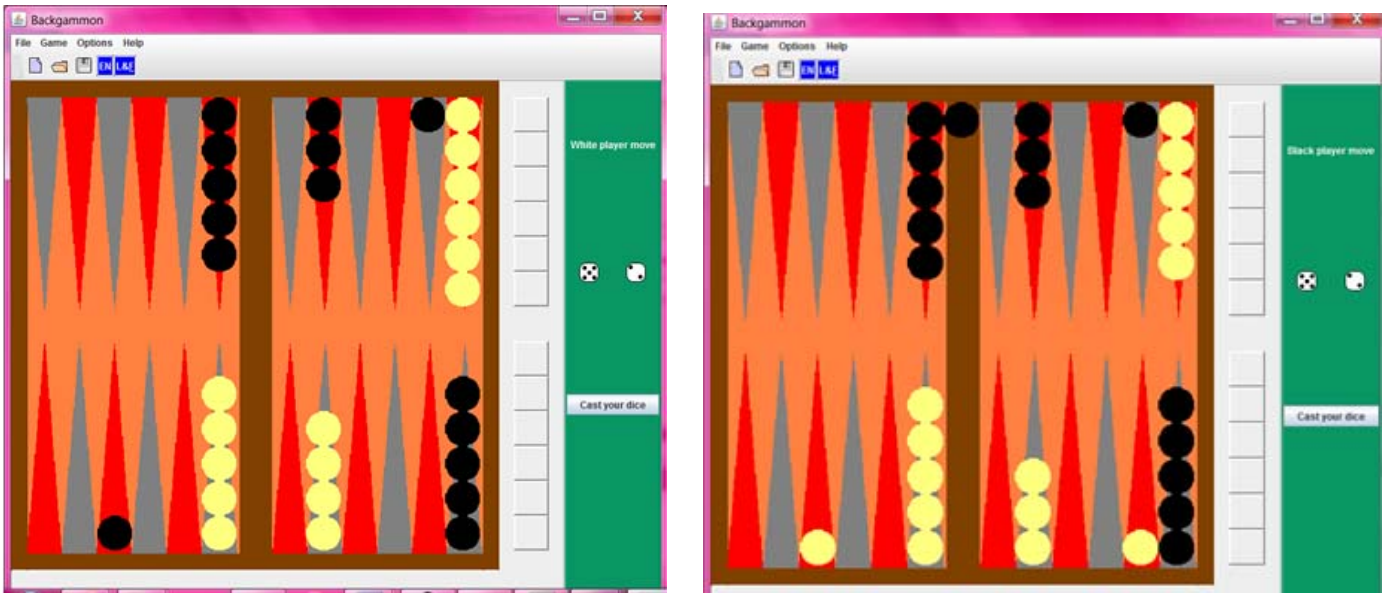

Fig. 7. A configuration of the backgammon board

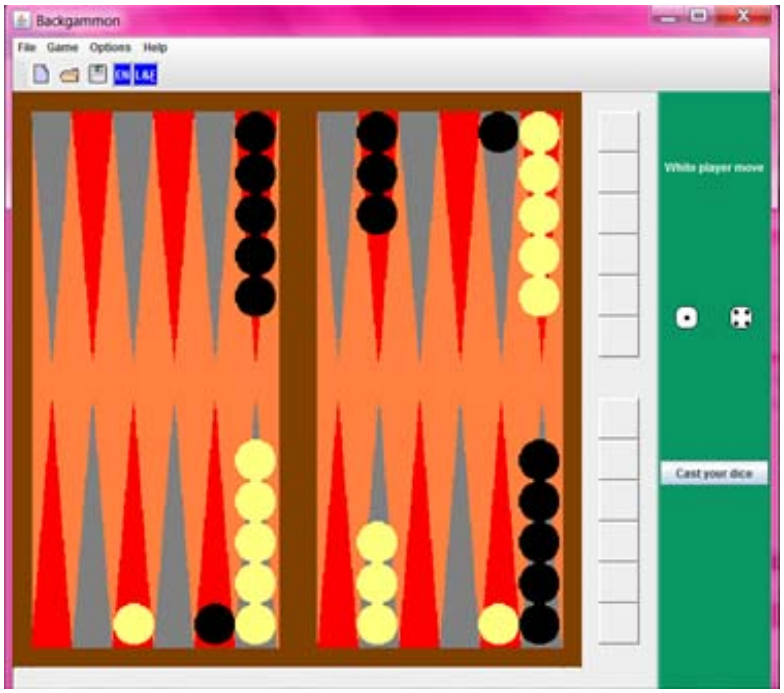

Fig. 8. A next configuration of the backgammon board for the fig 7.

Obviously, in case of computer user, there are many possible movements. In this case shown in figure 8, the maximum merit factor is assigned for that move in which the removed black piece is re-introduced in the game.

The software has several other options like selecting language for the menus, viewing or hiding toolbar, and selecting the aspect of the GUI. For example, in figure 9 are shown two available aspects. This was made using `UIManager` java class and `setLookAndFeel` method.

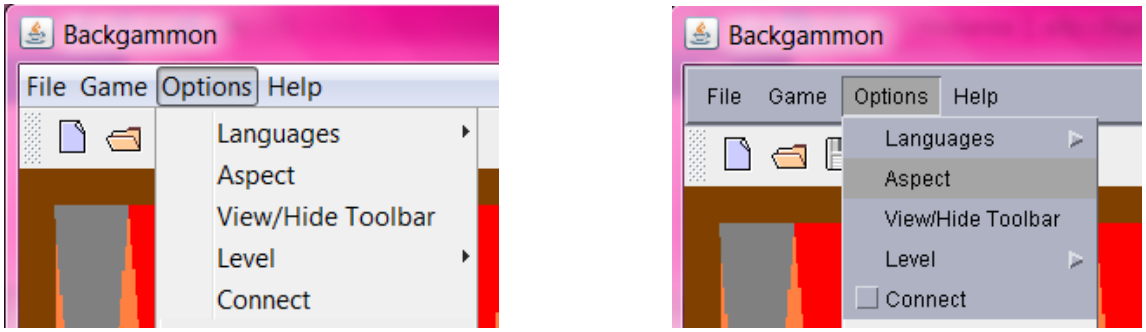

Fig. 1. The two aspects of the GUI

D. Algorithm for network playing

The application can by used also for playing in the network. For this purpose, it was used java package java.net. It was designed and implemented a class called "`ServerJoc`" allowing the connection of two classes "`Joc`" and thus the connection of two human users.

This class uses the ServerSocket and Socket classes to make possible the connection between two users. The communication is made using a port number. The server application has the graphical user interface shown in figure 10.

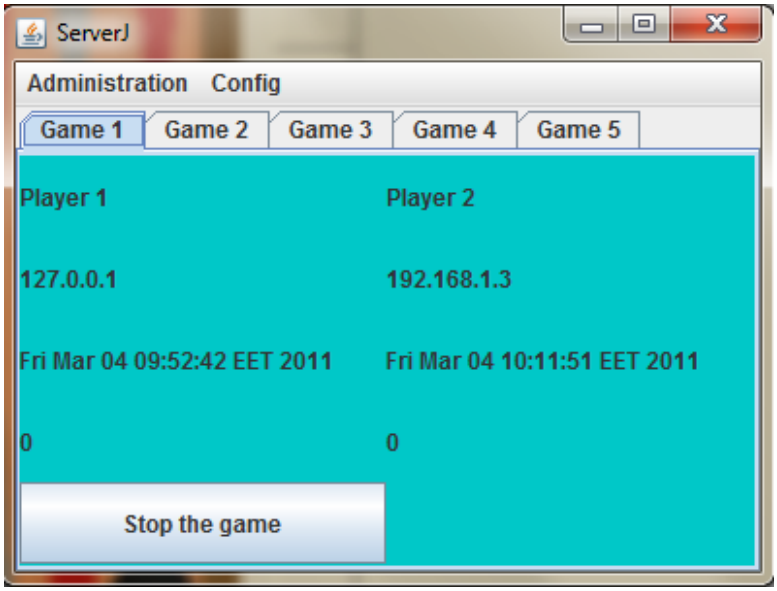

Fig. 10. Graphical user interface for the server application

This module was implemented for monitoring several games in the network. If users want to play in the network first of all the server application of the game must be started. The "Server" class contains a ServerSocket variable and a port number for performed client server socket communication.

For starting a game in the net, after the server application is started, the number of players must be two, selected from the "Game" menu of the previously software module, shown in figure 11. Then, from the "Option" menu, the IP address and the port number will be selected and using "Connect" option. Then the game will be started, like is also shown in figure 8.

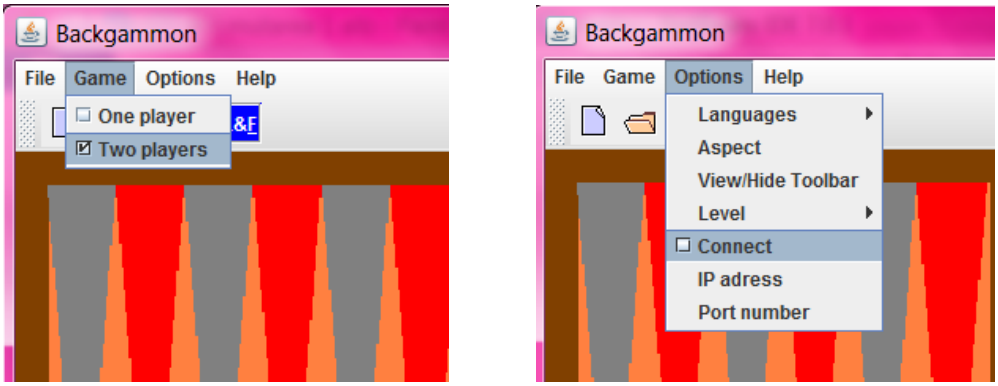

Fig. 11. Selecting the options for network play

III. Conclusions

Our application of new software for playing backgammon is intuitive and simple to use. It can be used for many purposes: first at all by the beginners to learn to play backgammon; then to play between two players in the network and to play versus computer. In the future we intend to improve the algorithm for playing versus computer using new intelligent algorithms.

References

[1] G. Tesauro. "Temporal Difference Learning and TD Gammon", *Communication of the ACM*, 38 (3), 1995

[2] G. Tesauro, Neurogammon: "A neural network backgammon learning program" , *Heuristic programming in Artificial Intelligence*, pag. 78-80, 1989

[3] Gerald Tesauro, "Programming backgammon using self-teaching neural nets", *Artificial Intelligence*, Elsevier Science, 2002

[4] Yaniv Azaria, Moshe Sipper, "GP-Gammon: Using Genetic Programming to Envolve Backgammon Players", *Lecture Notes in Computer Science*, pag 132-142, Volume 3447, 2005

[5] G. Tesauro, T.J. Sejnowski, "A Parallel Network that Learns to Play Backgammon", *Artificial Intelligence*, Elsevier Science, 1989

[6] http://www.no-gambling.com/book/computer-backgammon

[7] Hauk, T., Buro, M., Schaeffer, J., "*-Minimax Performance in Backgammon, Computers and Games" - *4th International Conference, CG'04*, Ramat-Gan, Israel, July 57, 2004, pp. 51-66

[8] http://en.wikipedia.org/wiki/Expectiminimax_tree

[9] Russell, S. J., Norvig, 2003. Articial intelligence: A modern approach, 2nd ed. Prentice Hall.

[10] Jonathan Schaefer, H. Jaap van den Herik, "Games, Computers and artificial intelligence", *Artificial intelligence 1*34, 2002, pp. 1-7.

[11] http://www.tinafad.com/dice2.php

[12] http://chandoo.org/wp/2008/08/13/simulate-dice-throws/