A Reversi Playing Artificial Intelligence Implementation

Ali Can ARIK
Email: acarik@gmail.com

Görkem KANDEMİR
Email: grkemkandemir@gmail.com

Hasan ATLI
h_atli@yahoo.com

***Abstract*—Reversi is one of the most popular board games. It is invented in 1883 and modern version of the game is known as Othello and is patented to Goro Hasegawa in 1970. This study consists of an artificial intelligence (AI) design to compete human being players. This paper is mainly composed of parts containing problem definition, algorithm design, implementation details, work distribution among group members and complete results.**

***Keywords*—Reversi, artifical intelligence, MiniMax algorithm, alpha-beta pruning, hash table.**

I. Introduction

In this study, a Reversi playing software is developed and implemented by exploiting the MiniMax and Alpa-Beta Pruning algorithms. The software is designed to play against human competitors. In this report; game definition, algorithmic and implementation details are provided as well as comparative performance evaluation results.

II. The Game

The game Reversi is believed to be invented in 1883. Othello is the name of the modern version of the game with slight changes of rules. Since Othello is a trademark, Reversi name can be used interchangeably.

Reversi is a two-player strategy game played on an 8x8 board. There are 64 identical game pieces called disks which are light on one side and dark in the other. Each of the disks' two sides corresponds to a player; however, any counters with distinctive faces are suitable. Players take turns placing the disks on the board with their assigned color facing up. During a play, disks of the opponent's color that are in a straight line bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The objective is to possess majority of the disks at the end of game.

For modern version of Reversi, rules state that the game begins with four disks placed in a square in the middle of the grid, two facing white side up and two with the dark side up, with same-colored disks on a diagonal with each other. Convention has initial board position such that the disks with dark side up are to the north-east and south-west from both players' perspectives. The initial state created in developed software in accordance with game rule convention is depicted in Fig. 1. Blue squares represent the possible available moves for the current player in turn.

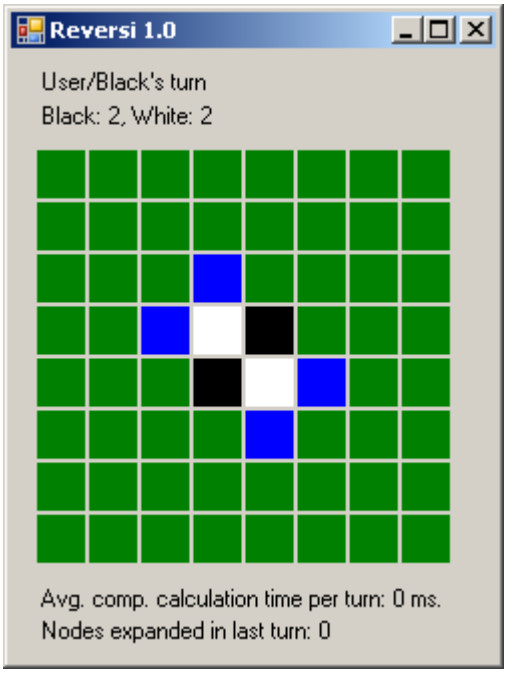

Fig. 1. Screenshot of the initial state in the software

III. Algorithmic Details

Details of the algorithm designed in this study are provided in this part. As it is described in Part II, Reversi is a two-player, zero-sum, discrete, finite, deterministic, and perfect information game.

In this study, Reversi is modeled as an adversarial game tree and then MiniMax algorithm is emplyed to maximize AIs gain. AIs gain is defined with an evaluation function, of which details are provided in Part III-A. In addition, Alpha-Beta Pruning method is used to decrease computational complexity as well as the number of expanded nodes during search.

In general, adversarial search refers to finding the optimal move by the player in turn, current player in other words. It is aimed to develop an AI to compete against a human player in this project. Therefore, the player in turn always refers to computer. Likewise, gain is defined in the AI player's perspective.

A. MiniMax Algorithm

MiniMax algorithm is based on finding the move which maximizes the AI players gain. On the other hand, the oppo-

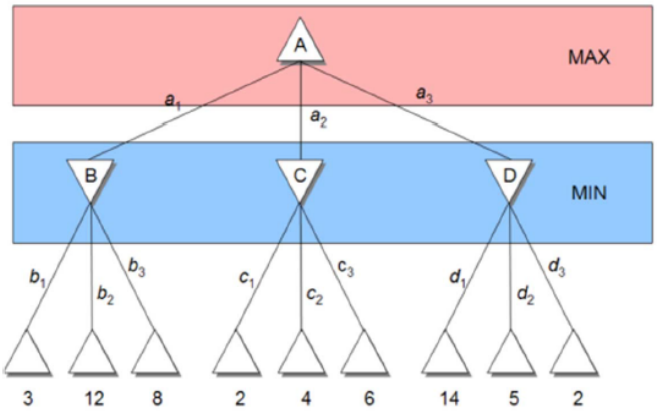

Fig. 2. Adversarial search tree for a move

nent is assumed to be a minimizing player that is he is assumed to select the moves minimizes the AI players gain. In other words, MiniMax algorithm assumes the opponent is trying to minimize AIs gain. Since the game Reversi is zero-sum, this assumption fits well, because one players gain is other players loss in our problem. Due to increasing branching factor, search depth must be limited. A comparative discussion about how search depth affects performance is provided in Part V. Adversarial search tree for a move (2 consecutive turns/plays) is depicted in Fig. 2.

B. Evaluation Function

As stated in Part III-A, search depth of the MiniMax algorithm is limited by a constant value. Therefore, MiniMax algorithm has to decide which move it should select by somehow comparing possible alternatives. In order to make this comparison, evaluation function of which details is used.

Evaluation function compares possible game states considering one or more criteria and assign a finite number to each game state. The criteria measures considered in evaluation function are very important in order to win the game, and they are game specific measures. The evaluation function value is a measure of how good a game state is. Based on this value, MiniMax algorithm tries to make rational decisions. Implemented evaluation function in this study is constructed as a weighted sum of three components and these components are calculated considering following strategies [2]:

- Maximum disc strategy
- Weighted square strategy (Positional importance/risks of disks)
- Maximum mobility strategy

As discussed in the preliminary report, these strategies have their advantages and drawbacks. Hence, relative weights of these components arranged in a way that drawbacks of these strategies are reduced and their advantages are maximized.

Maximum disc strategy component tends to maximize the difference in the number of discs of AI and its competitor in the favor of AI. However, this greedy approach is not useful at the early game since it causes loss of mobility. Therefore,

1								8
100	-25	10	5	5	10	-25	100	
-25	-25	2	2	2	2	-25	-25	
10	2	5	1	1	5	2	10	
5	2	1	2	2	1	2	5	
5	2	1	2	2	1	2	5	
10	2	5	1	1	5	2	10	
-25	-25	2	2	2	2	-25	-25	
100	-25	10	5	5	10	-25	100	
57								64

Fig. 3. Square weight values

weight of this component is kept at small at the early game and increased in the mid game.

Maximum mobility strategy component tends to maximize difference in the number of moves of AI and its competitor in the favor of AI. This strategy is especially important in the mid game. Therefore, its weight is increased linearly up to mid game and is constant from that point. Moreover, this components weight is larger than that of the maximum disc strategy component in order to overcome its greedy behaviour.

Weighted square strategy component tends to occupy strategically important squares in the game board and uses a look-up table which can be seen in Fig. 3 for evaluating the game state. This look-up table is a conventionally used square weight value table and widely used in scientific papers related to Reversi [3], [4], [5], [6]. Effect of this component is important at all stages of the game; therefore, its weight is selected in a way that this component dominates the output of evaluation function at all stages of the game.

In addition to these criteria, evaluation function also keeps track of evaluated game states by using a hash table in order to reduce computational complexity. Since there are 64 squares in the game board and each square can only take 3 different values, a key string consists of 64 bytes can uniquely define a game state in Reversi. Using this idea, whenever evaluation function is called, it calculates the unique key string for given game state and checks whether given state has been evaluated before or not. If not, the evaluation value is pushed into the hash table. A comparative evaluation of the effect of this approach on the computational complexity and computation time is provided in Part V.

C. Alpha-Beta Pruning

As it is described in Part III-A, the opponent is assumed to select the move which minimizes the players gain. By this assumption, some branches become unnecessary to investigate due to the fact that the opponent is not expected (assumed not to) select them. In other words, the opponent is not expected

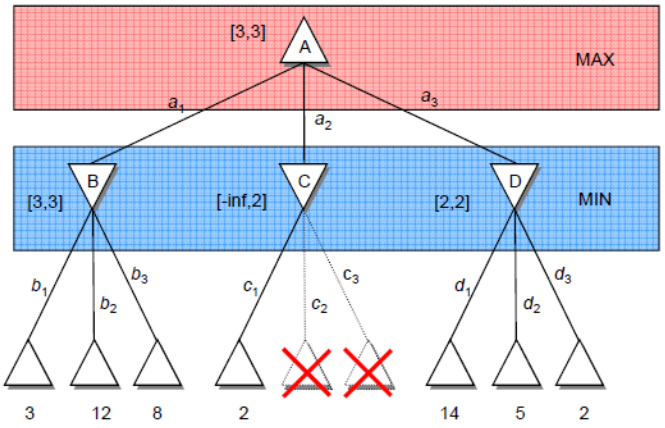

Fig. 4. Alpha-beta pruning example

to make a non-optimal move. Therefore, these branches may be pruned off from the game tree in order to decrease computational effort, and in order to exploit this fact alpha-beta pruning method is implemented together with the MiniMax algorithm.

In alpha-beta pruning method, leaf nodes are being evaluated with their evaluation function values. All of the branches descending from the first ancestor node among minimizing players choices are evaluated, then remaining choices of minimizing player are started to be evaluated. Any branch from remaining choices with smaller evaluation function value, namely gain, than the maximum of minimums obtained from branches evaluated up to current branch are pruned. These nodes may safely be pruned due to the fact that the opponent is assumed to be a minimizing player. An example game tree at which the alpha-beta pruning may be applied is provided in Fig. 4.

As a result, increase in computational complexity due to increase in branching factor may be suppressed.

IV. Implementation Details

Developed software is implemented in C# programming language and Microsoft Visual Studio 2008 is used as the choice of Integrated Development Environment. The choice of language and IDE are different from what was proposed in the preliminary report. We decided that C# is a more modern language and is suitable to our purposes with capabilities of easy interface design and class management. Also, the IDE of choice proved to have many advantages compared to what was planned earlier.

In order to visualize the game board, a 64 (8x8) square shaped text boxes are used. Their background colors are used to identify each cell as empty, white, black, or a possible move. After each move has been completed, the game board is updated in the graphical interface.

The game starts with a black move. After this, sides changes turns until the game ends, which occurs either there is no cell to move or two consecutive no-move states are detected.

In order to decide a move, a game tree is constructed from scratch every time. Each node in the tree has the successors which are the possible moves for the corresponding player. This operation is implemented using recursive methods, which implicitly ends up building the tree with depth-first greedy method. Once a leaf node is reached, which is either a state at the maximum depth relative to the current move or a node with no successor, the evaluation function is called for the corresponding game state. Alpha-beta pruning is integrated in tree traversal, resulting in a further increase in execution. Conventionally, the tree is constructed and the only benefit of pruning is avoiding evaluating unnecessary nodes. In this study, we do not even attach the unnecessary nodes to the game tree. This resulted in a further increase.

A game state is represented by a class with properties mainly: back pointer, 8x8 integer array. Further, it contains the necessary member methods and variables to support traversal among the adversarial game tree structure. It also includes many additional members in order to support every included software operation. The class diagram depicting all the classes used in the implementation is given in Fig. 5.

In addition, game state class has successors function to calculate possible moves and form the adversarial game tree.

V. Comparative Evaluation Results

The AI devised is evaluated by competing various expert players against it. Out of 42 games, the AI won 26 (62%), lost 11 (26%) games and 5 (12%) games ended in draw.

Alpha-beta pruning and hashing the evaluation function values in a look up table methods are implemented in order to make AI more efficient and fast.

In order to have an idea about the game performance, some internal variables of the game state are extracted and analyzed. The number of expanded nodes and the spent calculation time for move decision are selected as performance measures while judging Alpha-Beta Pruning and hashing methods.

Fig. 6 includes the plot of number of nodes expanded versus game turn and Fig. 7 includes the plot of processing time versus game turn number. As it is seen from these plots, various AI algorithm configurations are investigated and compared.

It can be deduced that alpha-beta pruning methods have significant effect on both execution time and number of expanded nodes. The worst AI performance in execution time sense occurs with alpha-beta pruning off. This is an expected result because alpha-beta pruning decreases the number of nodes expanded dramatically.

Since the evaluation function used in this study is very simple in the computation sense, the hashing function did not make execution time and expanded nodes any better. Using the hash table imposes some level of complexity on the software and it does not effectively increase performance for our case.

Another observation is that the branching factor of the game tree peaks at around the midway through the game. This is an expected result, as well. There is a strong correlation between number of possible moves and branching factor. Number of possible moves is relatively less at beginning of the game, because there are small number of placed tiles. However, as the number of placed tiles increases, number of possible

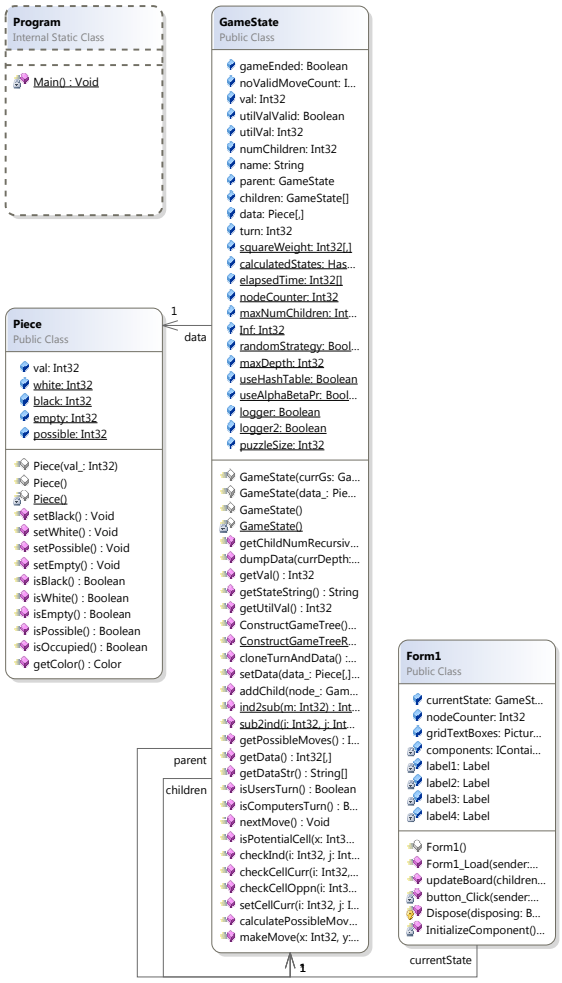

Fig. 5. Class diagram for the developed code

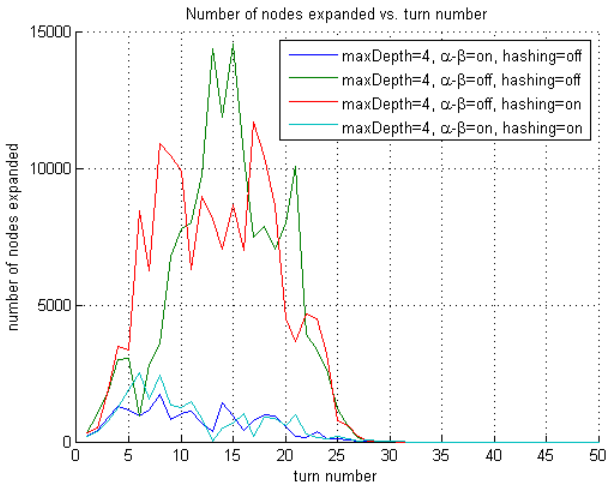

Fig. 6. Number of nodes expanded vs. turn number

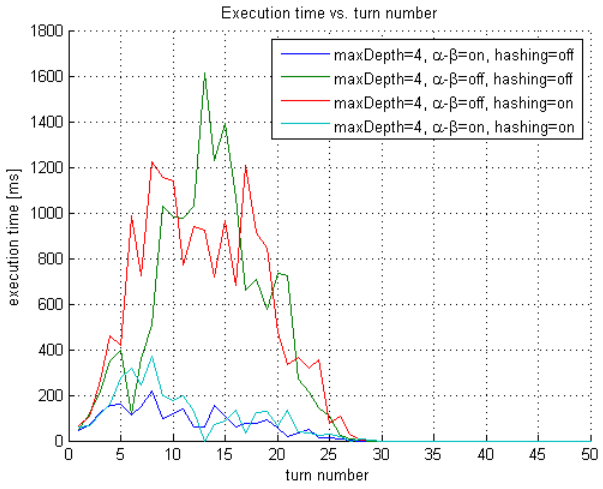

Fig. 7. Execution time vs. turn number

moves are increased; as a result, branching factor increases. At end-game, there are less empty places on the game board, therefore both number of possible moves and branching factor decreases. Variation of branching factor during the game may be observed from both Fig. 6 and Fig. 7.

Trial results with changing game configuration parameters alpha-beta pruning and hashing are provided in the figures given in the rest of this report. The staggering effect of alpha-beta pruning is easily seen for our case, in which branching factor is relatively high.

VI. Work Distribution

The work distributed among the team members as such:

- Ali Can ARIK: Development and implementation of the search problem as a game tree and the MiniMax algorithm, and Alpha-Beta Pruning method implementation.
- Görkem KANDEMİR: Development and implementation of the search problem as a game tree and the

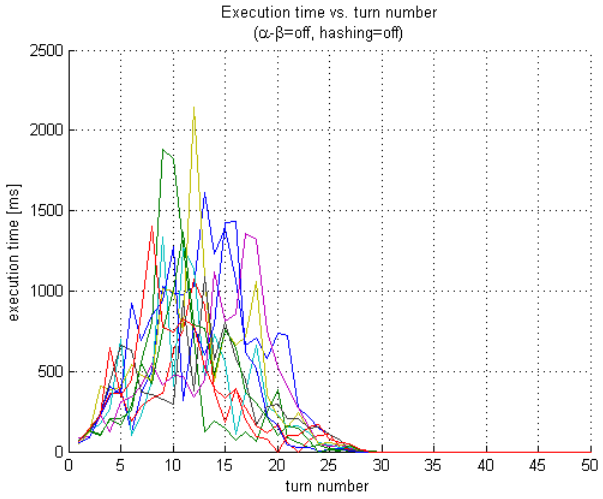

Fig. 8. Execution time versus turn number graph for maxDepth=4, ab=off, hash=off

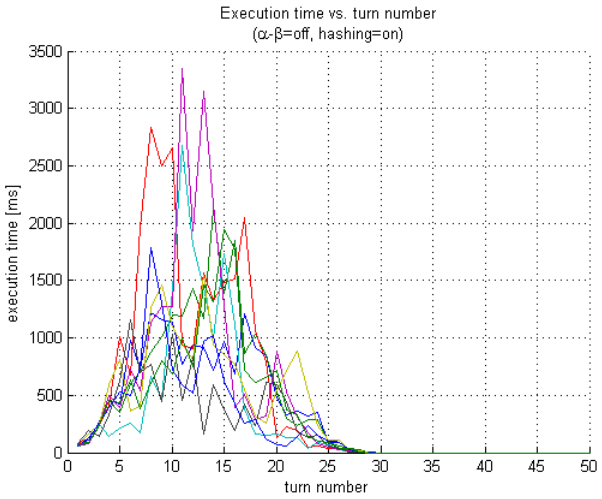

Fig. 9. Execution time versus turn number graph for maxDepth=4, ab=off, hash=on

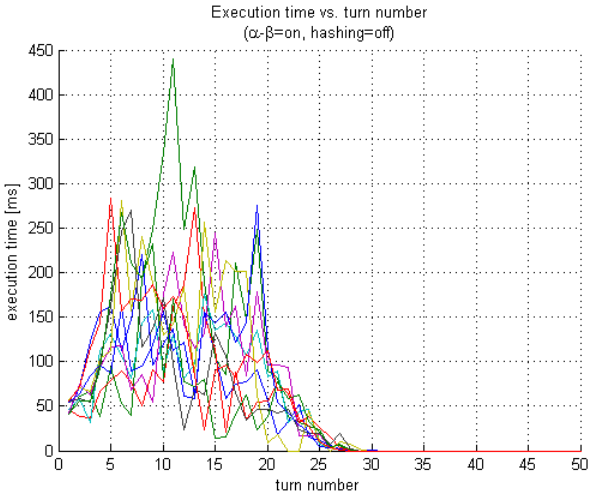

Fig. 10. Execution time versus turn number graph for maxDepth=4, ab=on, hash=off

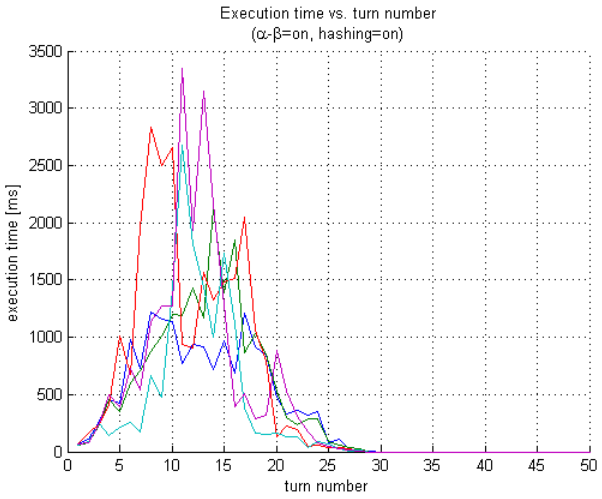

Fig. 11. Execution time versus turn number graph for maxDepth=4, ab=on, hash=on

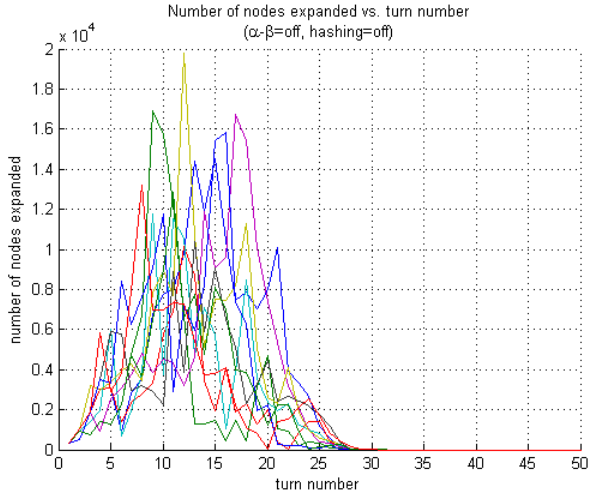

Fig. 12. Number of nodes expanded versus turn number graph for maxDepth=4, ab=off, hash=off

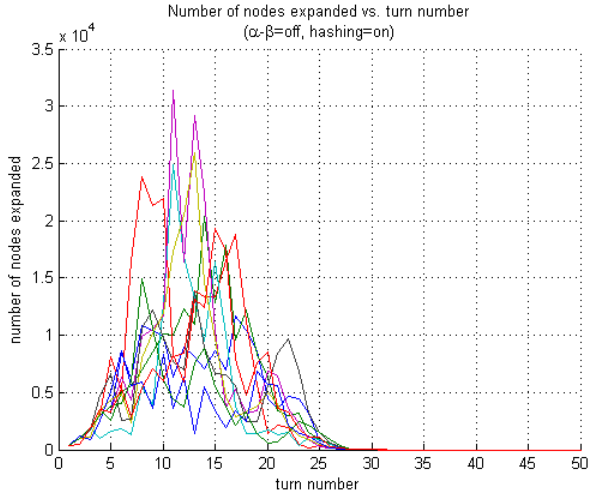

Fig. 13. Number of nodes expanded versus turn number graph for maxDepth=4, ab=off, hash=on

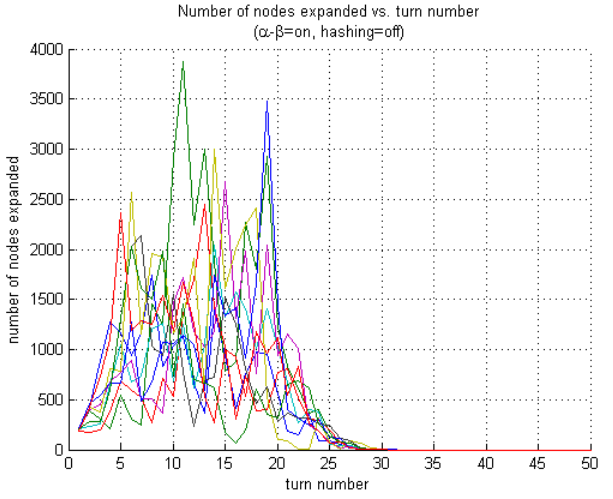

Fig. 14. Number of nodes expanded versus turn number graph for maxDepth=4, ab=on, hash=off

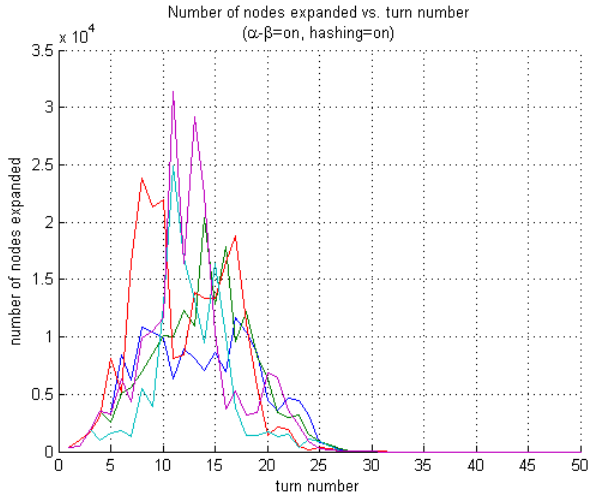

Fig. 15. Number of nodes expanded versus turn number graph for maxDepth=4, ab=on, hash=on

MiniMax algorithm; GUI design and implementation.

- Hasan ATLI: Development and implementation of the evaluation function; data structure and class abstraction.

VII. Conclusion

In this study, it is aimed to implement an artificial intelligence for the game Reversi. This paper constitutes the final report for the project. A general description of the game, algorithmic background and our complete approach to the problem along with the results are presented in this paper. Also, preliminary efforts in the selected programming framework are given. Results obtained using various algorithm parameters are presented.

The developed software can be seen to be a success, considering the results of the game to expert players. It proved itself as a worthy adversary as a Reversi player.

References

[1] Wikipedia contributors. Reversi. Wikipedia, The Free Encyclopedia. December 3, 2015, 16:36 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Reversi&oldid=693589941. Accessed December 3, 2015.

[2] Rosenbloom, Paul S. ”A world-championship-level Othello program.” Artificial Intelligence 19.3 (1982): 279-320.

[3] Lee, Kai-Fu, and Sanjoy Mahajan. ”BILL: a table-based, knowledge-intensive othello program.” (1986).

[4] Buro, Michael. ”Logistello: A strong learning othello program.” 19th Annual Conference Gesellschaft fr Klassifikation eV. 1995.

[5] Buro, Michael. ”The evolution of strong Othello programs.” Entertainment Computing. Springer US, 2003. 81-88.

[6] Binkley, Kevin J., Ken Seehart, and Masafumi Hagiwara. ”A study of artificial neural network architectures for Othello evaluation functions.” Information and Media Technologies 2.4 (2007): 1129-1139.