# 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.

# GAME DEFINITION

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 Figure 1. Blue squares mention the possible available moves for current player in turn.

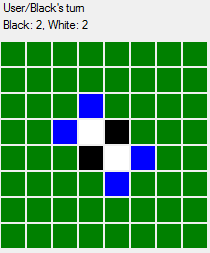


Figure 1: Initial State Created in the Developed Software

# ALGORITHMIC DETAILS

Details of the algorithm designed in this study are provided in this part. As it is described in Part 2, Reversi is a two players, 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 used to maximize AI’s gain. AI’s gain is defined with an evaluation function, of which details are provided in Part 3.1. In addition, Alpha-Beta Pruning method is used to decrease computational complexity as well as the number of expanded nodes during search.

## MiniMax Algorithm

MiniMax algorithm is based on finding move which is maximizing the AI player’s gain. On the other hand, the opponent is assumed to be a minimizing player that is he is assumed to select moves which is minimizing the AI player’s gain. In other words, MiniMax algorithm assumes the opponent is trying to minimize AI’s gain. Since the game Reversi is zero-sum, this assumption fits well, due to the fact that one players’ gain is other player’s loss in our problem. Due to increasing branching factor, search depth is limited to a constant value. A comparative discussion about how search depth affects performance is provided in Part 5. Adversarial search tree for a move (2 consecutive turns/plays) is shared in Figure 2.

## Evaluation Function

As stated in Part 3.1 search depth of the MiniMax algorithm is limited to 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.

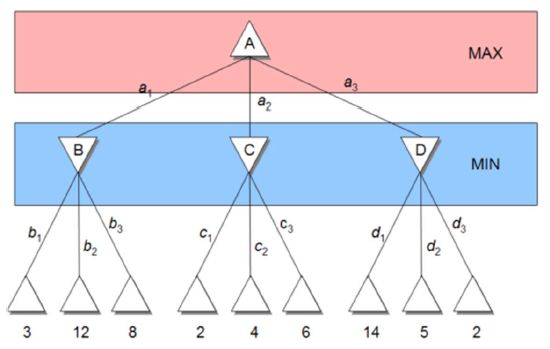


Figure 2: Adversarial Search Tree for a Move

Evaluation function compare possible game states considering one or more criteria and assign a finite number to each game state. The criteria measures which are 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 the game state is. Based on this value, MiniMax algorithm tries to make logical decisions.

Implemented evaluation function in this study is constructed as a weighted sum of three components and these components are calculated considering following strategies:

1. Maximum disc strategy
2. Weighted square strategy (Positional importance/risks of disks )
3. 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 fair of AI. However, this greedy approach is not useful at the early game since it causes loss of mobility. Therefore, weight of this component is very 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 fair of AI. This strategy is especially important in the mid game. Therefore, its weight is increased linearly up to mid game and get constant from that point. Moreover, this component’s weight is bigger than weight of maximum disc strategy component in order to overcome its greedy approach.

Weighted square strategy component tends to occupy strategic squares in the game board and uses a lookup table which can be seen in Figure 3 for evaluating the game state. This lookup table is a standard square weight value table and widely used in scientific papers related to Reversi. Effect of this component is important at all stages of the game; therefore, its weight is selected in a way that this component is dominates the output of evaluation function at all stages of the game.

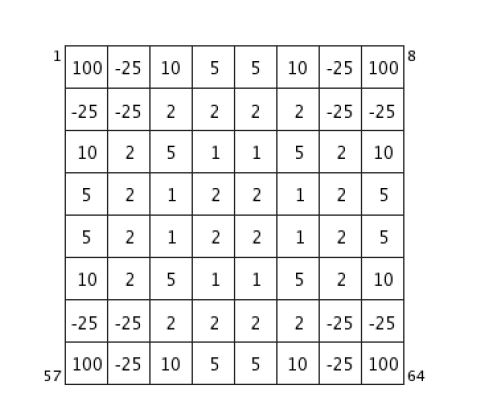


Figure 3: Square Weight Values

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 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 is evaluated before or not. A comparative evaluation of the effect of this approach on the computational complexity and computation time is provided in Part 5.

## Alpha-Beta Pruning

As it is described in Part 3.1, opponent is assumed to select the move which minimizes the player’s gain. By this assumption, some branches become unnecessary to investigate due to the fact that opponent is not expected (assumed to) select them. In other words, opponent is not expected to make a suboptimal move. Therefore, these branches may be pruned 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 started to be evaluated with their evaluation function values. All of the branches coming from first ancestor node among minimizing player’s 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 be safely pruned due to the fact that opponent is assumed to be a minimizing player. An example game tree at which the Alpha-Beta Pruning may be applied is provided in Figure 3.

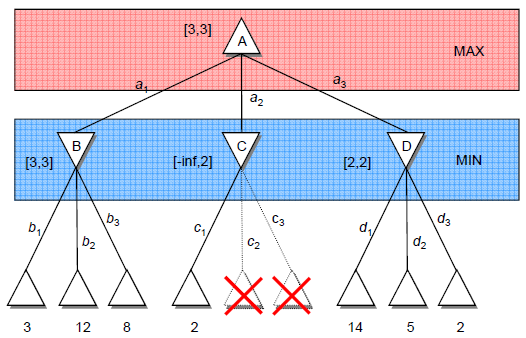


Figure 4: Alpha-Beta Pruning Example

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

# IMPLEMENTATION DETAILS

Developed software is implemented in C# programming language and Microsoft Visual Studio 2008 is used as an Integrated Development Environment.

In order to visualize the game board 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.

A game state is represented with a class with properties: back pointer, 8x8 integer array. In addition, game state class has successors function to calculate possible moves and form the adversarial game tree.

# Work LOAD DISTRIBUTION

The project work load is distributed among 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 MiniMax algorithm; GUI design and implementation.
* Hasan ATLI: Development and implementation of the evaluation function; data structure and class abstraction.

# COMPARATIVE EVALUATION RESULTS

The AI devised is evaluated by competing an expert player against it. The player never succeeded to win the game; 25% of games ended in a draw and the AI won 75% of the games.

In order to have an idea about the game performance, some internal variables of the game state are extracted and analyzed.

Fig1 includes the plot of number of nodes expanded versus game turn and Fig2 includes the plot of processing time versus game turn number. Provided plots include values for several AI configurations. A typical game scenario is taken to account.

Fig1:

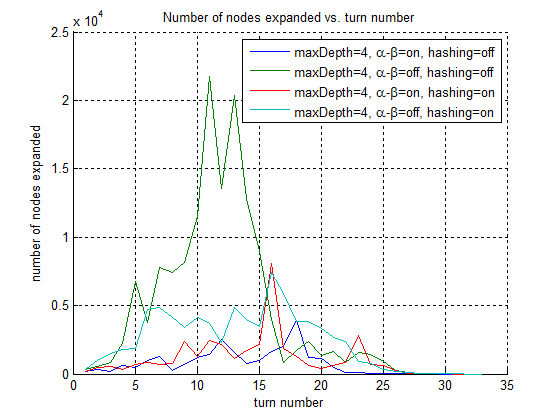
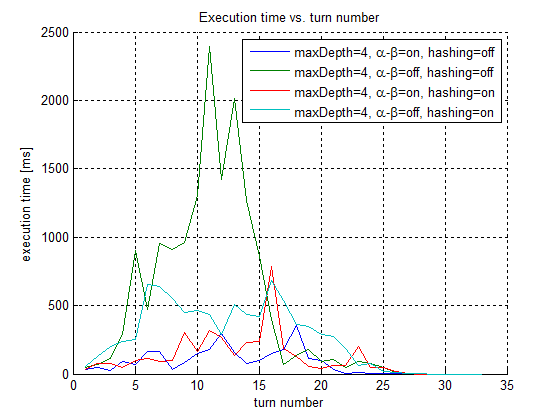


Fig2:



It can be deduced that hashing, alpha-beta pruning methods both increase execution. The worst scenario configuration in execution time sense occurs with both alpha-beta pruning and hashing are off. Another observation is that the branching factor of the game tree peaks at around the midway through the game. After this point, number of nodes expanded decreases significantly.

# CONCLUSION