The project I implemented is a chess game called Othello. Within my game, a user will move first as white and he or she will play against an intelligent bot with different strategies. The idea came from a homework in the AI course, where we were asked to write Python code for the game. I wanted to see how the execution time differs from running Python and Haskell. As a result, Haskell did outperform Python when given the same constraint to our AI. There are two versions of my code, one is the naive minimax algorithm and the other one is improved by using alpha-beta pruning. (see line 81-111 for the difference) One important note is that there is a slight modification of the rule in my game comparing the real one. The game will end immediately after one of the players does not have a legal move. In that way it is easier to determine the evaluation value for the Minimax algorithm which will be introduced below.

Minimax is a widely-used algorithm in two player turn-based games like Tic-Tac-Toe or different Chess games. The idea is that if we call the two players in the game *maximizer* and *minimizer* , we assume both of them are trying to play the game optimally and as a result *maximizer* will try to get the highest score possible while the *minimizer* will try to do the opposite and get the lowest score possible. With this in mind, one can construct a minimax tree with a desired depth and decides how to make the optimal move to reach the max/min value. Specifically in the Othello game, we can apply Minimax algorithm based on the value of difference between number of white pieces and black pieces. In that way, we know user will be aiming for the maximum value and our AI will aim for the opposite.

Alpha-beta pruning is a modified version of Minimax algorithm, in terms of decreasing the number of nodes evaluated by the minimax algorithm. The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. beta < alpha), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play. [1] In this way, the search time can be decreased since it will not explore some of the branches as they are guaranteed not to have the optimal value.

In terms of the performance, the alpha-beta pruning definitely beats the naive Minimax strategy. The responding time each round differs a lot when setting the tree depth to 6 (couple seconds vs. instant decision). For future work, I need to explore the parallelism potential for my project. I did find an article that suggests an approach of master and slave paradigm, which processors would divide work at each level of the tree. The section of the game that will be parallelized is the minimax function called by the master (the processor that starts the game). Inside this function, the master requests a number of slaves and then proceeds with the minimax function. [2] If I had more time, I definitely wanted to implement the parallelism and check the performance.

References:

[1] https://en.wikipedia.org/wiki/Alpha-beta_pruning

[2] http://www.pressibus.org/ataxx/autre/minimax/node6.html