Department of Computer Science

FAST-NUCES Lahore Campus

AI-2002 Artificial Intelligence

Team CyberPea – 19L-2347 && 19L-1246

Programming Assignment No 1 (Sections G)

Spring 2022 Assigned on 21/2/2022

First Deadline 28/02/2022 before 11:59 p.m.

Submission: Online on Google Classroom - Weight 3%

**Q1 – A detailed review of available Evaluation Functions (You must submit a report explaining various evaluation functions already available) [10]**

Claude Shannon proposed two types of chess programs [1]. Type A programs calculate all possible  
move sequences to a certain depth and returns the best possible move by using minimax. Since the average  
of available moves in the game of chess is around 35 Type A programs can only search up to a certain  
depth. Type B programs try to guess the important moves and make a deeper analysis by avoiding the  
moves that it finds inaccurate. This type of programs is able to calculate deeper since they don’t make an  
exhaustive analysis of the game tree. Type B programs only calculate the moves it thinks important and  
relevant to the position. Even though this type has the advantage of making a deeper analysis, overlooking  
a move that is actually important may cost losing a game completely. In the earlier eras of chess programs,  
type B computers dominated the field for a short period of time because of the lack of computing strength  
of earlier computers. Thus, type A programs was only able to search up to a quite small number of plies.  
A ply is a term used for denoting a move made by a player. A complete move in chess corresponds to a  
move by white and a reply by black side. Thus, a ply is a half-move.

After the improvements in the hardware of computers followed by the great increase in the computing  
speed of computers, type A programs began ruling the chess world. Increase in the computing strength  
made the usage of brute force method to exploit game trees possible. Current state of the art chess  
programs are well above the best chess players. Even the world’s best players acknowledge the fact that  
their laptops can beat them with a perfect score. The human vs computer chapter is over in the area of  
chess. Now chess players are considering chess programs as teachers rather than rivals.

Searching Techniques

One of two fundamental aspects of a chess program, Search exploits the game tree and try to find the best  
move sequence. This part is where much of the optimization algorithms are implemented. The most basic  
search algorithm is minimax where it checks all possible move sequences and return the move sequence  
that has the highest evaluation score assuming optimal play by the opponent. This approach is infeasible  
because of the very large number of available moves at each position. Therefore, several pruning  
algorithms are implemented to lower the branching factor of the game tree. Below is a list of search algorithms and search features that have been used previously to optimize the decision-making process.

* Alpha-Beta Pruning
* Principal Variation Search
* Transposition Table
* Killer Move Algorithm
* Quiescence Search
* Null move Pruning
* Futility Pruning
* Iterative Deepening

Alpha-Beta pruning, speeds up the search process by pruning the search tree without affecting the  
solution find [3]. The idea behind this algorithm is pretty simple. It assumes that if a move is too good to  
be true then it will not be played. To be more specific, alpha corresponds to the value that white player is  
guaranteed to receive for the current position. It is like a lower bound. Beta is the upper bound in the  
same sense that, any move with a value higher than beta will not be possible because black will simply  
choose the other path that results with a value of Beta [4]. So, a move is accepted only if it is between  
the boundaries alpha-beta. Depending on the side to move alpha and beta values are updated as the new  
moves are calculated. Pruning a branch because it has a lower value then alpha is called an alpha cut-off.  
Opposite is called beta cut-off.

Rahul [2] used Genetic Algorithms to tune the parameters of the evaluation function. In this paper the  
dependence of the current top-level chess programs to brute-force search is criticized and implementing  
a self-improving chess program is pointed out as a solution. Genetic Algorithms are used to evolve the  
parameters of the evaluation function.

Coevolution is used where the initially randomly created players(chromosomes) play against each  
other. 15 positional parameters are considered apart from material values to evaluate a chess board. These  
positional parameters are chosen with respect to their importance and their applicability on Positional  
Value Tables (PVTs) / Heat Maps. The evaluation function of the chess program is implemented as PVTs.  
A PVT holds a value for all possible 64 squares on a chess board for each piece. Thus intuitively,  
a PVT for a knight will hold high values for centre squares and low values for corner and side squares,  
because knights are usually stronger on the centre squares. This idea of holding a value table for each  
piece enables the program to learn on its own by evolving the values stored in a PVT.

**Q2 – A detailed description of your own evaluation function [10]**

The following pseudocode describes the overview of the steps involved in the decision-making process.

PAWN 10

KNIGHT 30

BISHOP 30

ROOK 50

QUEEN 90

KING 900

1. DECIDE MOVE.
   1. **RECURSIVELY RUN MINIMAX** 
      1. COMPUTE POSSIBLE ACTIONS
      2. IF DEPTH ZERO OR CHECKMATE
         1. EVALUATE SCORE OF BOARD AT LEAF NODES
         2. **EXIT RECURSION / BASE CASE** (GO TO 1b.)
      3. ELSE
         1. BACKUP *ACTION LIST* OF CURRENT STATE
         2. FOR EACH PLAYER (ALTERNATELY)
            1. BACKUP BOARD STATE
            2. SELECT MOVE FROM *ACTION LIST*
            3. APPLY THE SELECTED MOVE IN b.
            4. CHANGE PLAYER
            5. DEPTH= DEPTH -1
            6. **RECURSIVELY RUN MINIMAX.** (GO BACK TO 1a)
   2. **AFTER EVALUATING A LEAF NODE / EXIT RECURSION**
      1. UNDO APPLIED MOVE.
      2. RESTORE ACTION LIST.
      3. CALCULATE MIN/MAX SCORE FOR WHITE/BLACK RESPECTIVELY
      4. SAVE INDEX OF BEST MOVE AS CALCULATED IN 1b(iii)
      5. RETURN BEST MOVE.
2. APPLY BEST MOVE.

Bibliography

[1] Vazquez Fernandez. An evolutionary algorithm coupled with the Hooke-Jeeves algorithm for tuning  
a chess evaluation function. IEEE Congress on Evolutionary Computation, 2012

[2] Rahul A R. Phoenix: A self-optimizing chess engine. International Conference on Computational  
Intelligence and Communication Networks, 2015.

[3] Boskovic b. A differential evolution for the tuning of a chess evaluation function. 2006 IEEE  
International Conference on Evolutionary Computation, 2006.

[4] Cook Andrew. Using chunking to optimise an alpha-beta search. International Conference on  
Technologies and Applications of Artificial Intelligence, 2010.