Department of Computer Science

FAST-NUCES Lahore Campus

AI-2002 Artificial Intelligence

Team CyberPea – 19L-2347 && 19L-1246
Programming Assignment # 1 (Section G)
Spring 2022 Assigned on 21/2/2022
First Deadline Sunday 06/03/2022 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 can 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 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 chess. Now chess players are considering chess programs as teachers rather than rivals.

The evaluation functions available online made use of mainly two techniques. The first technique focused on giving a specific weightage to the specific pieces which is guided by the chess knowledge and the number of moves a piece can move. Then, to calculate the total material value of a board, all these weights are summed up. For example, King was given the maximum weightage out of all other pieces due to its importance in the game, the Queen is given the second highest weightage because it is an extremely versatile and invaluable piece. This way every piece was assigned a weightage and the pieces of opponent were given the same magnitudes but with opposite sign so that the game remains a zero-sum game. Until a player captures another player's piece, this evaluation function would keep returning 0 score for every possible action which was its drawback as well. It would only work well after a piece has been captured. My artificial intelligence instructor made use of a random function which will select any of the possible moves until a piece was captured which did not actually solve the problem at hand. This technique is mainly referred to as "Material Score Evaluation".

The second technique focused on the positional aspects of the game rather than material aspect. Overtime, Chess players over the world have developed several rules of thumb which a good chess player is expected to cater to maximize their probability of winning the game. Examples of such rules are:

- King should always be kept safe by placing your other pieces around it.
- Knights should not be placed in corners which will limit their possible moves.
- Try to control the center of the board.
- Do not lock in Bishops
- Try to keep your pawn structure intact.

There are many more such techniques used to achieve positional dominance over the opponent. Positional dominance can allow a player with lesser material score to be able to win the game. This evaluation technique is implemented widely using heat map arrays which contains a set of neutral, positive and negatives values for each piece which eventually guides us about the advantage of a possible move. Furthermore, this kind of evaluation can also be applied to the initial stage of the game when no pieces have been captured yet (this was an area where the material evaluation function would fail because the sum initially would be 0).

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

CyberPea's evaluation function employed both the materialistic technique as well as positional technique to find a best possible move at a given point in a game of chess. We defined the following weightages to respective pieces to evaluate materialistic score:

```
#define PAWN 10
#define KNIGHT 30
#define BISHOP 30
#define ROOK 50
#define QUEEN 90
#define KING 1000
#define CHECK 1000
```

Secondly, we deployed heat map arrays to evaluate the positional advantage of a move which also relies on the guiding principles provided above in explanation of available position evaluation functions. Now, at the start of the game when material evaluation score is 0. CyberPea's evaluation function will resort to evaluation all heat map arrays in order to achieve positional dominance in the game. The arrays implemented were as follows:

```
float blackPawn [8][8] =
        {
            {7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0},
            {5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0},
            {1.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 1.0},
            {0.5, 0.5, 1.0, 2.5, 2.5, 1.0, 0.5, 0.5},
            {1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0},
            {0.5, -0.5, -1.0, 0.0, 0.0, -1.0, -0.5, 0.5},
            {0.5, 1.0, 1.0, -2.0, -2.0, 1.0, 1.0, 0.5},
            {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
        };

float whitePawn [8][8] =
        {
            {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
            {0.5, 1.0, 1.0, -2.0, -2.0, 1.0, 1.0, 0.5},
            {0.5, -0.5, -1.0, 0.0, 0.0, -1.0, -0.5, 0.5},
            {1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0},
            {0.5, 0.5, 1.0, 2.5, 2.5, 1.0, 0.5, 0.5},
            {1.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 1.0},
            {5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0},
            {7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0}
        };

float Knight [8][8] =
        {
            {-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0},
            {-4.0, -2.0, 0.0, 0.0, 0.0, 0.0, -2.0, -4.0},
            {-3.0, 0.0, 1.0, 1.5, 1.5, 1.0, 0.0, -3.0},
            {-3.0, 0.5, 1.5, 2.0, 2.0, 1.5, 0.5, -3.0},
            {-3.0, 0.0, 1.5, 2.0, 2.0, 1.5, 0.0, -3.0},
            {-3.0, 0.5, 1.0, 1.5, 1.5, 1.0, 0.5, -3.0},
            {-4.0, -2.0, 0.0, 0.5, 0.5, 0.0, -2.0, -4.0},
            {-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0}
        };
```

```
float blackBishop [8][8] =
    {
        { -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0},
        {-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0},
        {-1.0, 0.0, 0.5, 1.0, 1.0, 0.5, 0.0, -1.0},
        {-1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 0.5, -1.0},
        {-1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, -1.0},
        {-1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0},
        {-1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5, -1.0},
        {-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0}
    };

float whiteBishop [8][8] =
    {
        {-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0},
        {-1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5, -1.0},
        {-1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0},
        {-1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, -1.0},
        {-1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 0.5, -1.0},
        {-1.0, 0.0, 0.5, 1.0, 1.0, 0.5, 0.0, -1.0},
        {-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0},
        { -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0}
    };

float blackRook [8][8] =
    {
        {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
        {0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5},
        {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
        {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
        {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
        {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
        {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
        {0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0}
    };
```

```
float whiteRook [8][8] =
        {
            {0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0},
            {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
            {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
            {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
            {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
            {-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5},
            {0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5},
            { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}

        };

float Queen [8][8] =
        {
            { -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0},
            {-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0},
            {-1.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0},
            {-0.5, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5},
            {0.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5},
            {-1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0},
            {-1.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, -1.0},
            {-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0}
        };
```

```
float blacking [8][8] =
      {
          { -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0},
          {-1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0},
          {2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0},
          {2.0, 3.0, 1.0, 0.0, 0.0, 1.0, 3.0, 2.0}
      };

float whiteKing [8][8] =
      {
          {2.0, 3.0, 1.0, 0.0, 0.0, 1.0, 3.0, 2.0},
          {2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0},
          {-1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0},
          {-2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          {-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0},
          { -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0}
      };
```

Our evaluation function ran each time the AI tried out a possible move and the pseudocode for it is provided below:

```
Define score = 0
Iterate through the board
{
|       At every position [r][c] check the piece and its color (e.g., Black Bishop)
|       {
|       |       If (piece == white)
|       |       {
|       |       |       Score += BISHOP // add respective weight if piece white
|       |       |       Score += whiteBishop[r][c] // add respective positional weight
|       |       }

|       |       Else
|       |       {
|       |       |       Score -= BISHOP // subtract respective weight if piece black
|       |       |       Score -= blackBishop[r][c] // subtract respective positional weight
|       |       }
|       }
}

If (White Checks Black)
        Score += CHECK // if black under check, then this is a good move for white (add score)
Else
        Score -= CHECK // if white under check, then this is a good move for black (minus score)

Return score
```

This returned score was then fed into our minimax + alpha beta pruning function and the scores for each move were evaluated against each other, ultimately leading to the selection of the 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