Programming project

Artificial Intelligence

The project has 4 parts. In each part you are asked to write two programs. The programming language should be JAVA, Python, C/C++.

Part I: MINIMAX (45%)

Write two programs that get as input two file names for input and output board positions, and the depth of the tree that needs to be searched. The programs print a board position after White plays its best move, as determined by a MINIMAX search tree of the given depth and the static estimation function given in the Morris-Variant handout. That board position should also be written into the output file. In addition, the programs prints the number of positions evaluated by the static estimation function and the MINIMAX estimate for that move. The board position is given by a list of 22 letters. See the Morris-Variant handout for additional information.

First program: MiniMaxOpening

The first program plays a move in the opening phase of the game. We request that you name it **MiniMaxOpening**.

For example, the input can be:

(you type:)

MiniMaxOpening board1.txt board2.txt 2

(the program replies:)

Board Position: xxxxxxxWxxWxxxBxxxxxx
Positions evaluated by static estimation: 9.
MINIMAX estimate: 9987.

Here it is assumed that the file board1.txt exists and its content is:

xxxxxxxWxxxxxxBxxxxxx

The file board2.txt is created by the program, and its content is:

xxxxxxxWxxWxxxBxxxxxx

(The position and the numbers above are most likely correct. They are given just to illustrate the format.)

Please use the move generator and the static estimation function for the opening phase. You are not asked to verify that the position is, indeed, an opening position. You may also assume that this game never goes into the midgame phase.

Second program: MiniMaxGame

The second program plays in the midgame/endgame phase. We request that you call it **MiniMaxGame**. For example, the input can be:

(you type:)

MiniMaxGame board3.txt board4.txt 3

(the program replies:)

Board Position: xxxxxxWWWxWWxBBBxxxxx.
Positions evaluated by static estimation: 125.
MINIMAX estimate: 9987.

Here it is assumed that the file board3.txt exists and its content is:

xxxxxxxxWWxWWxBBBxxxxx

The file board4.txt is created by the program, and its content is:

xxxxxxWWWxWWxBBBBxxxxx

(The position and the numbers above may not be correct. They are given just to illustrate the format.)

Part II: ALPHA-BETA (35%)

In this part you are asked to write two program that behave exactly the same as the program of Part I, but implement the ALPHA-BETA pruning algorithm instead of the MINIMAX. Notice that these programs should return the exact same estimate values as the programs of Part I; the main difference is in the number of nodes that were evaluated. We request that you call these programs **ABOpening** and **ABGame**.

Part III: PLAY A GAME FOR BLACK(10%)

Write the same programs as in Part I, but the computed move should be Black's move instead of White's move. We request that you call these programs **MiniMaxOpeningBlack** and **MiniMaxGameBlack**.

Part IV: STATIC ESTIMATION (10%)

Write an improved static estimation function. The new function should be better than the one which was suggested in the handout. Rewrite the programs of Part I with your improved static estimation function. We request that you call these programs **MiniMaxOpeningImproved** and **MiniMaxGameImproved**.

Due date: to be announced.

For C/C++ implementations and other special cases it may be necessary for you to be present when your project is being tested.

What you need to submit:

Submit a documented source code, and, if relevant, executables. This should include the source for all eight programs.

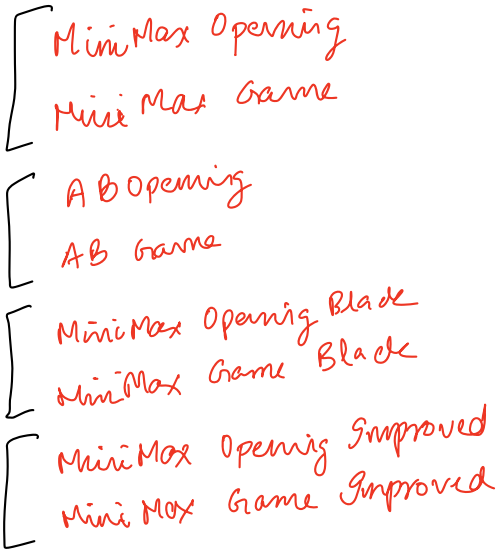
Show examples of the program output when applied to several positions. Give at least two cases in which alpha-beta produces savings over MINIMAX.

Show at least two examples where your static evaluation function produces different moves than the standard evaluation function. Write a short (one or two paragraphs) explanation of why you believe your function to be an improvement over the function proposed by the instructor.

The submission should be a single zip file named with your net ID. For example, if your net ID is xyz1234 your submission file should be named xyz1234.zip.

Community Standards and Conduct

This is an individual project. You may discuss with other students the performance of your program, but you are not allowed to share code. All programs will be tested for plagiarism.



Recursive Minimax

Minimax is used to dynamically evaluate a position in the game. The procedure is described here as a depth first search. The nodes in the search tree are positions. A node is a "**max**" node if it describes a position where the "maximizer" is to make a move. It is a "**min**" node if it describes a position where the "minimizer" is to make a move. The children of a node are all the possible positions that can be reached after one move. With each position x we associate the value V_x. The function **static**(x) gives a static evaluation of the position. The following two recursive procedures evaluate "dynamically" the value V_x of the node x:

if x is a max node: $V_x = \text{MaxMin}(x)$, where:

$$\text{MaxMin}(x) = \begin{cases} \text{static}(x) & \text{if } x \text{ is a leaf} \\ \max\limits_{\text{children } y \text{ of } x} \text{MinMax}(y) & \text{otherwise} \end{cases}$$

if x is a min node: $V_x = \text{MinMax}(x)$, where:

$$\text{MinMax}(x) = \begin{cases} \text{static}(x) & \text{if } x \text{ is a leaf} \\ \min\limits_{\text{children } y \text{ of } x} \text{MaxMin}(y) & \text{otherwise} \end{cases}$$

These routines can also be written as:

```
MaxMin(x):
if x is a leaf return static(x).
else {
    set v = −∞
    for each child y of x:
    v = max(v, MinMax(y))
    return v
}
```

```
MinMax(x):
if x is a leaf return static(x).
else {
    set v = +∞
    for each child y of x:
        v = min(v, MaxMin(y))
    return v
}
```

Recursive Alpha-Beta pruning

For each node x retain the range of values of its parents so that: $\alpha \leq V_{\text{parents of x}} \leq \beta$. If the root of the tree is a MAX node it is evaluated by: MaxMin(root, $-\infty, +\infty$).

MaxMin(x, α, β)**:**

1. if x is a leaf return static(x).

2. else do steps 2.1, 2.2, 2.3

2.1. set $v = -\infty$

2.2 for each child y of x:

2.2.1 $v = \max(v, \text{MinMax}(y, \alpha, \beta))$

2.2.2 if($v \geq \beta$) return v

2.2.3 else $\alpha = \max(v, \alpha)$

2.3. return v

MinMax(x, α, β)**:**

3. if x is a leaf return static(x).

4. else do steps 2.1, 2.2, 2.3

4.1 set $v = +\infty$

4.2 for each child y of x:

4.2.1 $v = \min(v, \text{MaxMin}(y, \alpha, \beta))$

4.2.2 if($v \leq \alpha$) return v

4.2.3 else $\beta = \min(v, \beta)$

4.3 return v

Notes:

. Step 2.2.2 is a β cut. The value returned at 2.2.2 will not affect the outcome since it will never be the minimum at 4.2.1.

. Same observation holds for the α cut at Step 4.2.2.