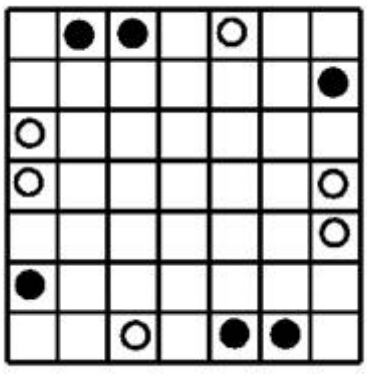
2. copying parts of the solutions to questions for Part II from other students

If you are uncertain about any of these guidelines, please discuss with your instructor as soon as possible.

Introduction

This modified version of connect-4 is played on a 7x7 grid as follows: Starting from the initial position illustrated below, players take turns moving one piece of their colour by a number of squares as explained below, either horizontally or vertically. No jumping is allowed. White plays first. Pieces can only move to unoccupied squares. The winner is the first player to form a 2x2 square of four pieces. In addition, pieces create an "impedance" field in the immediate (8-squares) neighbourhood around them, affecting the number of squares an opponent piece can move. The following table describes the number of squares a piece can move given the number of opponent pieces in its neighbourhood:

Number of opponent pieces in the surrounding 8 squares	Number of squares the piece can move
0	[1-3]
1	[1-2]
2	1
3 or more	0 (pinned)



starting board state

Part I

Implement a game-playing agent to play the game of Dynamic Connect-4 against a human. Your agent must be capable of playing either white or black. The time limit for the

computer (which must be one of the Trotttier Engineering Linux servers, accessible via 156TRLinux.ece.mcgill.ca) to transmit its move to the game server is 10 seconds. Although this is not mandatory, we suggest that you display a simple text-only "visualization" of the game state, both for debugging purposes, but also, for human inspection of the game play sequence. Such a display should represent the current board state as a matrix of comma-separated characters, with O denoting a white piece, X denoting a black piece, and suitably formatted whitespace denoting an empty square. For example, the starting board configuration above would be represented as follows:

```
 ,x,x, ,o, ,
 , , , , , ,x
o, , , , , ,
o, , , , , ,o
 , , , , , ,o
x, , , , , ,
 , ,o, ,x,x,
```

Each square in the grid can be labeled by its <x,y>position, with <0,0> referring to the top-left corner of the board (thus, <2,1> corresponds to the top-left black piece in the initial formation). Plays are communicated to the game sever by specifying the position of the piece to move in the form <x,y>followed by one of the compass directions, N, S, E, W; and the number of squares to move. For example, to move the black piece at the top left of the board one square to the left, the command would be 21W1. Each time it performs a move, your agent should also echo the associated command for human inspection.

For this part of the assignment you may evaluate game states as being either a win (+1), a loss for (-1), or a neutral state (0). Implement both the minimax and alpha-beta algorithms. Also, your program should have an option for specifying the initial game state (either through an input file or manually, as you wish). This will be used to test your program with specific game scenarios.

In your report, consider the three game states shown below:

```
 ,o, , ,x,x,x        , , , , ,o,        , , , , ,o,o
 , , , ,x,o,x        , , , , ,x,         , , , ,o,x,x
 , , , , , ,         , , , ,x,x,         , , , , , ,
o, , , , , ,         , , , , ,o,         , , , , ,o,
 , , , , , ,         , , , , ,x,         , , , , , ,
 , , , , , ,o        ,o, , , ,o,         ,x, , , ,x,o
o,o, , , , ,x       x,x,o,o, , ,         ,x,x, , , ,o
     (a)                 (b)                 (c)
```

1. Indicate the total number of states visited by your program, starting from each state shown, when using depth cutoffs of 3, 4, 5 and 6, both with minimax and alpha-beta. Assume it is white's turn to play.

2. Does the number of states visited depend on the order in which you generate new states during the search? 1) Explain your answer and 2) justify it using results from your program.
3. If two agents are to play against each other for scenario (c) for a depth cutoff of 6, for which, if any, of minimax and/or alpha-beta pruning will the losing agent try to delay its defeat? Explain your answer.

Part II

In Part I of this assignment, our evaluation function was extremely simplistic. This left our agent with the ability to make intelligent moves only when its lookahead horizon could detect a sequence leading to a guaranteed win. However, as described in the text, when we are forced to cut off the search at non-leaf nodes, we can exploit a heuristic evaluation function to provide a meaningful value representing the "goodness" of the corresponding state.

Your task is to design a useful heuristic evaluation function for non-terminal nodes and demonstrate that it leads to improved performance (compared to the evaluation function in Part I). Note that in addition to playing "well" when confronted with non-terminal nodes, your agent should also exhibit the following behaviour:

- when victory is certain for your agent, it should try to win as soon as possible
- when defeat is certain for your agent, it should play so as to delay its defeat as much as possible.

In your report:

1. Provide a rationale for the choice of the improved evaluation function you used. In particular, explain how you implemented the behaviour described above.
2. For a given search depth, does your improved evaluation function reduce the average number of nodes visited with 1) minimax? 2) alpha-beta? Illustrate appropriately.
3. Discuss the computational tradeoffs with the use of a more complex evaluation function with respect to the depth of the game tree that can be evaluated.
4. For a depth cutoff of 4 for scenarios (a) and (c) in part 1, give a log of the game by two agents that use your developed heuristic. (In other words, run two instances of the code, with each playing one of the sides, and having its moves communicated via the game server, just as will be done in the class tournament).