Hassan Mehdi

09/27/2016

Test Plan

Reason for test	Input	Expected Output	Actual Output
Prove the king can jump over an enemy piece	1 1 1 0 0 1 1	the number of jumps is 1	the number of jumps is 1
Prove the king can jump over several enemy pieces	1 1 2 0 0 1 1 3 3	the number of jumps is 2	the number of jumps is 2
Prove the program can take several test cases	2 1 1 0 0 1 1 1 2 0 0 1 1 3 3	the number of jumps is 1 the number of jumps is 2	the number of jumps is 1 the number of jumps is 2
Prove that the king can jump in any direction, as well as make smart decisions on which way to jump	1 1 6 0 0 1 1 3 3 5 5 5 3 5 1 3 1	the number of jumps is 5	the number of jumps is 5

Description of Problem

This program will allow you to create an instance of a checker board, placing down a king piece, followed by your normal pieces, followed by the placement of your opponent's pieces, which your king will attempt to eliminate as many as possible.

To create an instance(s) of the board you would first type the number of boards you wish to create, and entering that value.

Then you would type how many of your pieces you will have on the board (The first piece will always be your king). Followed by a space, and then the number of your opponent's pieces that will be on the board, and pressing enter.

You will then type the coordinates of your king piece by entering its row (from bottom – 0 – to top – 7), followed by a space, and then entering its column (from left – 0 – to right – 7), followed by pressing enter

You will then do the same for each of your non-king pieces, followed by doing the same for each of your opponents pieces.

After you finish, the program will move your king piece in a branching path to decide which path will yield the most number of jumps, and then will return that number of jumps as the value of the function.

EX:

1 - number of instances

1 1 - number of your checker, number of opponents checker

3 3 - coordinates of your checker

4 4 - coordinates of opponents checker

Pre/Post Conditions

int Reset()

Precondition – none

Postcondition – each value in the board array will be given an ‘e’ for empty, signifying a piece can occupy it later on

Void Addchecker()

Precondition – int yourcheckers, int oppcheckers, and positioning of each checker must be set

Postcondition – The position of each checker will be recorded and set into the board array

Int numjumps()

Precondition – int yourcheckers, int oppcheckers, and positioning of each checker must be set in order for the function to find the maximum number of jumps possible

Postcondition – the maximum number of jumps possible will be recorded and displayed

Void print()

Preconditions – the coordinates of each checker in the board array must be known

Postconditions – the board will be printed out on an 8x8 grid showing the location of your king, any pieces you have, and your opponent’s pieces