Quinn Roemer

CISP - 430

Assignment 3

2/15/2018

Program 3.0 - Maze Implementation

Description:

The goal for this assignment was to finish some code that my professor gave me. The code was supposed to solve a maze written as a two-dimensional array. The function that we wrote had to use a recursive solution and had to be placed inside of the professors code. Please note, the source code is rather lengthy and will take up several pages.

Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer

#include <iostream>
using namespace std;

#define SIZE 10

//Creating a class to hold the position/direction on the maze.
class Cell {
public:
   int row = 0;
   int col = 0;
   int dir = 0;
};

//Creating an array to hold the solution.
Cell sol[SIZE*SIZE];

//Defining the maze the computer has to solve.
int maze[SIZE][SIZE] = {
   1,1,1,1,1,1,1,1,1,1,
   1,0,0,0,1,1,0,0,0,1,
   1,0,1,0,0,1,0,1,0,1,
   1,0,1,1,0,0,0,1,0,1,
   1,0,0,1,1,1,1,0,0,1,
   1,1,0,0,0,0,1,0,1,1,
   1,0,0,1,1,0,0,0,0,1,
   1,0,1,0,0,0,0,1,1,1,
   1,0,0,0,1,0,0,0,0,1,
   1,1,1,1,1,1,1,1,1,1
};

//Function Prototypes.
void build(int);
void printSolution(int);
```

```
int cellok(int);
int getNextCell(int);

//Main function to execute.
int main()
{
   //Setting start points for the computer.
   sol[0].row = 1;
   sol[0].col = 1;
   sol[0].dir = 0;

   //Calling build.
   build(0);
}

//Build function, written by Quinn Roemer.
void build(int n)
{
   //Initializing a variable as a loop counter.
   int p = 0;

   //Line used to debug.
   //cout << sol[n].row << " and " << sol[n].col << endl;

   //This iterates until p is 4 or greater.
   while (p < 4)
   {
            //Calling getNextCell() and setting temp to its return.
            int temp = getNextCell(n);

            //If temp is zero the function will return.
            if (temp == 0)
            {
                     return;
            }

            //Otherwise, cellok() is called.
            else
            {
                     if (cellok(n))
                     {
                              //If cellok() returns true then the program enters direct recursion.
                              build(n + 1);
                     }
            }
            //If the computer is at cell (8, 8) it will print the solution.
            if (sol[n].row == 8 && sol[n].col == 8)
            {
```

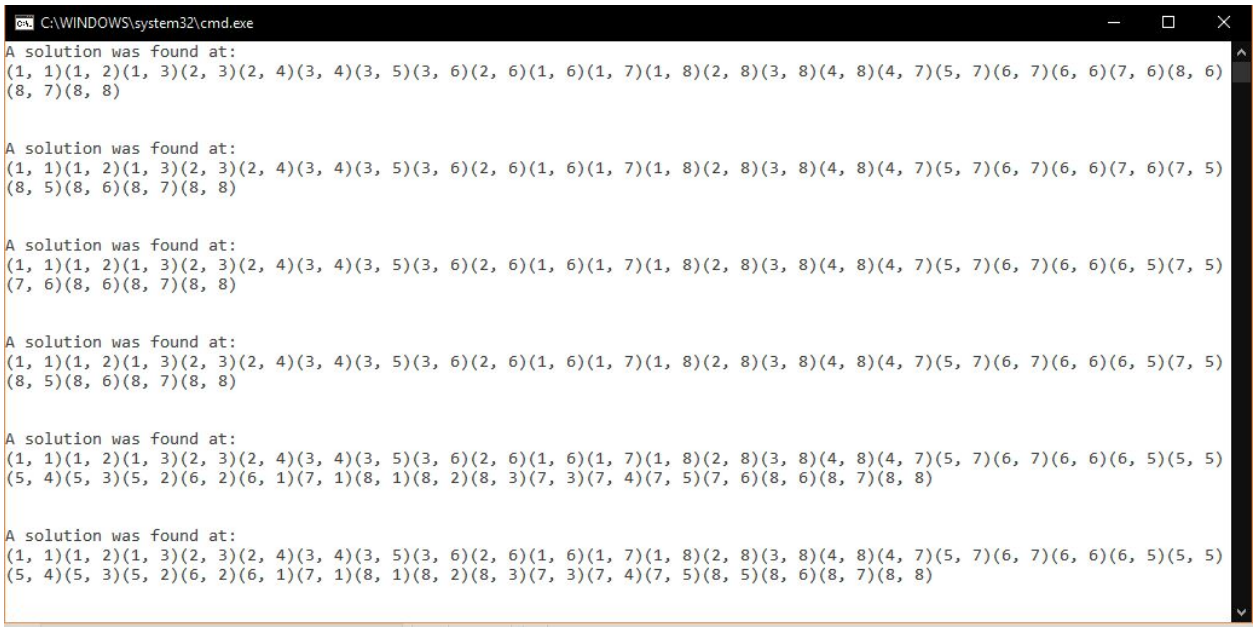
```
                printSolution(n);

                //Calling return to find another solution.
                return;
        }

        //Incrementing the loop counter.
        p++;
    }
}

//This function prints the current solution held in the sol array.
void printSolution(int n)
{
    int i;
    cout << "\nA solution was found at:\n";
    for (i = 0; i <= n; i++)
    {
        cout << "(" << sol[i].row << ", " << sol[i].col << ")";
    }
    cout << endl << endl;
}

//This function sets the current position/direction of the computer on the maze.
int getNextCell(int n)
{

    //Set initial position and direction for the next cell.
    sol[n + 1].row = sol[n].row;
    sol[n + 1].col = sol[n].col;
    sol[n + 1].dir = 0;

    //Try all positions; east, south, west, north.
    //Increment direction of current cell.
    //Increment position of next cell.

    switch (sol[n].dir)
    {
    case 0:
        sol[n].dir = 'e';
        sol[n + 1].col++;
        return 1;

    case 'e':
        sol[n].dir = 's';
        sol[n + 1].row++;
        return 1;
```

```
   case 's':
            sol[n].dir = 'w';
            sol[n + 1].col--;
            return 1;

   case 'w':
            sol[n].dir = 'n';
            sol[n + 1].row--;
            return 1;

   case 'n':
            return 0;
   }

   return 0;
}

//This function checks to see if the computer is on a wall or a place where
//it has already travelled.
int cellok(int n)
{
   int i;

   if (maze[sol[n + 1].row][sol[n + 1].col] == 1)
   {
            return 0;
   }

   for (i = 0; i < n; i++)
   {
            if (sol[n + 1].row == sol[i].row && sol[n + 1].col == sol[i].col)
            {
                     return 0;
            }
   }
   return 1;
}
```

Output:

Note: Since the output is large it will take place over two images.

(1 of 2)



(2 of 2)

My Function:

The function that I wrote is actually quite simple. All it does is call the getNextCell function that the professor wrote to determine the computer's position and direction. If that function returns zero the function itself returns. However, if it does not return zero the function calls the cellok function to determine whether or not the move is valid. If it is, it calls the next instance of build via direct recursion. If it is not, it calls getNextCell again to look at another possible move. It does this a maximum of four times before the function “gives up” and returns to the function that called it. Below is a look at my functions source code pulled out of the main source code.

My Functions Source Code:

```
//Build function, written by Quinn Roemer.
void build(int n)
{
  //Intilizing a variable as a loop counter.
  int p = 0;

  //Line used to debugg.
  //cout << sol[n].row << " and " << sol[n].col << endl;

  //This iterates until p is 4 or greater.
  while (p < 4)
  {
        //Calling getNextCell() and setting temp to its return.
        int temp = getNextCell(n);

        //If temp is zero the function will return.
        if (temp == 0)
        {
              return;
        }

        //Otherwise, cellok() is called.
        else
        {
              if (cellok(n))
              {
                    //If cellok() returns true then the programs enters direct recursion.
                    build(n + 1);
              }
        }
        //If the computer is at cell (8, 8) then it will print the solution.
        if (sol[n].row == 8 && sol[n].col == 8)
        {
              printSolution(n);
```

```
                //Calling return to find another solution.
                return;
            }

            //Incrementing the loop counter.
            p++;
    }
}
```

Conclusion

This assignment was an interesting one. I had never really programmed a recursive function before this. I had always strayed from them deeming them complex and confusing. This assignment helped me understand the concept of recursion better. However, I am still an iteration man!