**Visual Recursion Maze Generator**

**Created By:**

**Jonathan Sullivan**

**Table of Contents**

1. **General Description page 3**
2. **How to Use the Software page 4**
3. **Source Code page 16**
4. **Journal/Blog page 76**
5. **Resume page 80**
6. **Philosophy page 81**
7. **Career Plans page 82**
8. **Sources page 82**

**General Description**

For the final class of the Computer Science major at St. Norbert College we are given a semester long project that we are to work on. This project can be a continuation of a past project or a completely new one. For this capstone course we are not only required to work on the project itself by we also had to create a website in which to show our progress, present what we completed in the last few weeks of the semester, and talk with the professors in order to go over our design decisions for the project and why we did things the way we did.

For this project I was tasked with creating a maze generator that could be used by future Computer Science Students in order to help learn recursion. The maze generator would need to show a secondary program moving through the maze. For this I decided to create the maze generator in c# due to the many visual methods that could be implemented. I then created an API in which to connect a program to the maze generator. While this was implemented in c# I designed the secondary test programs in c++ since that is the language that is used currently to when teaching computer science students about recursion. This is the current language at the time that this manual was created in May of 2015. Below is a copy of the requirements that I received at the beginning of the semester for the Visual Recursion project. The description was created by Dr. Dave Pankratz.

Description:

Students usually understand the basic ideas involved in recursion algorithms but have problems writing code that implements these ideas. It would be great if recursive code could be visualized so students could actually “see” exactly what their programs are doing.

Requirements:

1. Design a “maze creator” module with the ability to open and save different mazes.
2. Design an API for the maze so that any program has the ability to communicate its location within the maze.
3. Make the maze visually show current and visited locations along with its state (blocked/unblocked)
4. Develop a maze UI that has controls: animate, speed, step, undo, redo, reset, etc.
5. Construct several programs to run on the maze, including instruction type programs that demonstrate depth first and breadth first recursive algorithms.
6. Consider developing a game that uses the maze or an enhanced version of the maze.

**How to Use the Software**

**Basic Information:**

 The c# maze generator was created using Microsoft Visual Studio c# 2008 express edition with a .NET framework of 3.5. These were also used to create the dll that was created for the API. The c++ test programs were created using Microsoft Visual Studio c++ 2008 express edition.

**The dll Function Library:**

 Before either of the programs can communicate however the c++ program must be created. While most of the c++ program can be created using the common functions, however all of the interaction between the c# maze generator which will be called the client from now on and the c++ program which will be known as the server must use the API functions. These functions include several for movement, checking information about the maze, and connection. This part of the User Manual will go through each of the functions that are available for use and how they can be called.

 **StartConnection():** This function will need to be called by any program that wants to connect to the client. It will start the connection between the programs from the server side. It will wait for a connection for about two minutes once called.

 **EndConnections():** This function is the pair to the StartConnection(), instead of starting the connection it will close it. This should be called only after the use of the client is finished as there is currently no way to reestablish connection unless both programs are restarted.

 **MoveSetLocation(int row, int column):** This function can be used to move to a specific place within the maze. It will move to the row and column indicated by the integers that are passed by parameters. This function also has the added capability of mimicking a recursive calls return process since it can be called whenever the recursive function returns. An example of this can be seen in the Example Server Program that is provided. It is located on Page INSERT PAGE.

 **Move Functions:** These are a series of four functions MoveUp(), MoveDown(), MoveLeft(), MoveRight(). Each of these will move the client cursor in the maze one square in the direction indicated. It will not allow movement outside of the maze but will allow movement onto walls.

 **CheckPath Functions:** These are a series of four functions CheckPathUp(long \* result), CheckPathDown(long \* result), CheckPathLeft(long \* result), and CheckPathRight(long \* result). Each of these functions will check the path one square in the indicated direction in order to see whether or not the square is a wall, outside of the maze, or a path and whether or not the path has been traversed. If it detects a wall it will return a zero, it will return a -1 if it is outside of the maze, a one if it is a path, and a two if it is a path that has already been traversed. For example if you are checking up to see if there is a path you would call CheckPathUp(result) which would return a 1 if there is a path. These functions will receive the results as long values which will be stored in a long pointer upon return. Due to this a long pointer will need to be used as a reference parameter in order to obtain the result. For example you would need to declare a pointer long \* result; and then call the function CheckPathUp(result);. To get the data you would use \*result in order to dereference it.

 **CheckWall Functions:** These are a series of four functions that will check in the direction listed to see if there is a wall or not one square in the direction. These functions are CheckWallUp(long \* result), CheckWallDown(long \* result), CheckWallLeft(long \* result), and CheckWallRight(long \* result). These will return -1 if the square is outside of the maze, zero if there is a wall, or a one if there is a path. These functions will receive the results as long values which will be stored in a long pointer upon return. Due to this a long pointer will need to be used as a reference parameter in order to obtain the result. For example you would need to declare a pointer long \* result; and then call the function CheckWallUp(result);. To get the data you would use \*result in order to dereference it.

 **CheckEnd(long \* result):** This function will check to see if the cursor has reached the end of the maze. It will return a one if it has reached the end or a zero if it has not. This function will receive the results as long values which will be stored in a long pointer upon return. Due to this a long pointer will need to be used as a reference parameter in order to obtain the result. For example you would need to declare a pointer long \* result; and then call the function CheckEnd(result);. To get the data you would use \*result in order to dereference it.

 **GetCurRow(long \* result):** This function will obtain the current row that the client cursor is in. It will receive the result as a long value which is stored in a long pointer. For example you would need to declare a pointer long \* result; and then call the function GetCurRow(result);. To get the data you would use \*result in order to dereference it.

 **GetCurCol(long \* result):** This function will obtain the current column that the client cursor is in. It will receive the result as a long value which is stored in a long pointer. For example you would need to declare a pointer long \* result; and then call the function GetCurCol(result);. To get the data you would use \*result in order to dereference it.

These functions are all stored in the MazeFunctionLibrary dll. This will need to be included in the server program. This can be done by adding

#import "..\MazeFunctionLibrary\bin\Release\MazeFunctionLibrary.tlb" raw\_interfaces\_only

using namespace MazeFunctionLibrary;

Outside of the main function with all of the other #includes and namespaces. You would then need to create an instance of the library which can be done by using

HRESULT hr = CoInitialize(NULL);

rmgLibraryPtr rmgPtr(\_\_uuidof(ManagedClass));

To call a function you would then need to use the name of the insance that was created. For the example above a function could be called from rmgPtr. To call the function the arrow operator -> is used. So for the example above you could call MoveUp() by using rmgPtr->MoveUp();.

**The Maze Generator Client:**

 The Maze Generator Client Opens with a simple introduction page that give basic information about the program.

 The opening page

Each menu item has information under it that can be used except for the edit which has features that have not yet been implemented. The Help menu is used to display information about the project and a help screen with basic usage information. These next few screen shots are of the main help form.



   

The about form will give basic information that does not really pertaint to the use of the software.



The open file will simply open a standard open file window which the user can then choose which maze to open. A maze uses the file type .rmg and is the only type of file that the maze generator can recognize. The New menu item however will open a different form which will start the creation of a new maze. Below is a screen shot of the starting form for maze creation.



It will simply ask the user for a number of rows and columns for the maze, currently the number of rows and columns is limited to between two and seven. If cancel is clicked then you simply return to the previous form. If OK is clicked however you move to the next step of creation where you actually create the maze.



On the right side it gives a simple instruction for the creation process and at first displays the given maze with all gray boxes to signify that only walls are created. By clicking on one of he squares it changes color in the order displayed on the right. Below is an exaple of a 3x3 maze that can be created.



The light gray squares signify a path while the yellow shows the start and the blue shows the end. Once completed the user can save the maze which opens the general save window. As said above all mazes must be saved with the extension .rmg. Once created a mazze is not automatically opened. Rather you must go into the open menu item in order to open it.

Below is an example of an opened maze that was previousely created.



7

6

2

5

3

4

1

1. Past Moves: This will display any past moves that the user has made. It will list where they moved from and where they moved to. Along with whether the square is a path or a wall.
2. Establish Connection Button: This button is what is used to establish the connection with the server program. Once clicked there are two minutes available for the server to connect to the maze generator client.
3. Current Position Marker: This marks the current position within the maze.
4. Finish Line: This square denotes the end of the maze.
5. Auto Move Control: This checkbox will start out unchecked which will require the user to click the manual movement button in order to move through the maze. If checked it will automatically read the instructions from the server and move through the maze as soon as the information is received.
6. Manual Movement Button: This is required for when the auto move is disabled. It will allow step by step movement through the maze by reading the instructions from the server.
7. Move Cursor: This along with the row and column text boxes will allow the cursor to move to a specific location.

**Establishing a Connection:**

 A connection is established by both the server and the client programs. While they do both connect to the same place they each require the user or the server program to start them. For the client the connection can be established only when a maze has been loaded. It is started by clicking the start connection button. For the server program it is started once the StartConnection() function has ran. Both the server and the client will wait for two minutes for the connection before moving on.

**Sending Commands (The Pipes):**

 The movement through the maze is all done with the dll functions that the server uses. After a connection has been made all other dll functions can be called to move and obtain information about the maze. The dll fuctions for not directly influence the maze. Rather they send short command codes to the client which then reads them and completes the command. The commands are sent through a pipe which connectes the two programs enableing them to communicate. Below is a diagram of what the pipes look like.

Server Write

Server Read

Client Read

Client Write

**C++ Server Program**

**C# Maze Generator Client**

The above diagram gives a simple view of how the two programs are connected. The overall set up uses two pipes to connect the programs. The first is allows data to be sent from the client to the server and is used primarily to send data back to the server. The data that is sent are the results of the functions that return information. These functions are the check walls, check paths, check end, and get row and column functions. The other pipe is for the server to send data to the client. This is used by all of the functions in the dll since each one sends a command code to the client in order to move through the maze. An example of the pipe can be seen with the GetCurRow(result) function. The function will send the command code to the client via the server write pipe which is then read by the client. The client will then obtain the current row and send the information through the client write pipe which is then read by the server into the result variable.

**Using the client software:**

 While most of the code maze is traversed through the use of the dll functions there are some features in the client that allow the user to control the maze. The first being the abliity to move to a specific location in the maze using the move cursor button and the row and column text boxes. This can be used as a way to test the maze by starting in a different spot. The other is the auto move check box and the manual movement button. These will change whether the maze will read the instructions from the pipe connection automatically or if it will require the user to press the manual movement button in order to step through the maze.

**Other Aspects:**

 The client program and the dll do have a few bugs that might have been missed. The primary being that when the connection of one program is closed the other crashes. This works fine if it is closed only when closeing both of the applications since there would be no other work that would need to be completed.

 **DLL Creation:**

 As said above the dll was created using c# and then registered using the command prompt provided by the visual studio 2008. For the creation I followed a tutorial from <https://support.microsoft.com/en-us/kb/828736>. For this rather than create the key as the tutorial shows I opened the opened properties of the project and added it there. The location can be seen in the screenshot below. I then went into the the visual studio 2008 command prompt where I wnet to the directory where the project .dll was located and then entered the command RegAsm.exe ManagedDLL.dll /tlb:ManagedDLL.tlb /codebase which created the .tlb which is used to include the library in the c++ program.



**Source Code**

**MazeFunction Library:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.IO.Pipes;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*NOTES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* - Each function will need to write to the file that the

 \* Recursive Maze will read.

 \* - The pipe name is rmgPipe

 \* - MUST CALL StartConnection() BEFORE any other functions

 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace MazeFunctionLibrary

{

 public interface rmgLibrary

 {

 //int Add(int Number1, int Number2);

 //Will need basic functions for movement

 //AND for setting up the pipes

 void StartConnection();

 //void StartReadConnection();

 void EndConnections();

 void MoveSetLocation(int row, int column);

 void MoveUp();

 void MoveDown();

 void MoveRight();

 void MoveLeft();

 int CheckWall(int direction);

 int CheckPathUp();

 int CheckPathDown();

 int CheckPathLeft();

 int CheckPathRight();

 int CheckWallUp();

 int CheckWallDown();

 int CheckWallLeft();

 int CheckWallRight();

 int CheckEnd();

 int GetCurRow();

 int GetCurCol();

 };

 //Interface Implementation

 public class ManagedClass : rmgLibrary

 {

 //Set up pipe to send data to the client

 const string PIPESENDING = "rmgPipeServer";

 System.IO.Pipes.NamedPipeServerStream pipeStreamWrite = new NamedPipeServerStream(PIPESENDING);

 //StreamWriter sw = new StreamWriter(pipeStreamWrite);

 //Set up pipe to receive data from the client

 const string PIPEREADING = "rmgPipeClient";

 System.IO.Pipes.NamedPipeClientStream pipeStreamRead = new NamedPipeClientStream(PIPEREADING);

 //public int Add(int Number1, int Number2)

 //{

 // return Number1 + Number2;

 //}

 //Connection Fucntions

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StartWriteConnection\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be used to create the Connections to the pipes. It will

 // first create the writing pipe and then wait for connection before it connects

 // to the reading pipe. This function is required to create the connecion between

 // the maze and the server.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void StartConnection()

 {

 //Create the named pipe to send data

 // using (pipeStreamWrite)

 // {

 //PUT TRY CATCH HERE

 Console.WriteLine("[Server] Waiting for client connection");

 pipeStreamWrite.WaitForConnection();

 //pipe has been connected

 Console.WriteLine("[Server] Write Connection Established");

 // }

 pipeStreamRead.Connect(120000); //Wait for up to two minutes to connect.

 if (pipeStreamRead.IsConnected == false)

 {

 Console.WriteLine("[Server] Unable to connect Read Pipe to the client");

 }

 else

 {

 Console.WriteLine("[Server] Read Connection Established.");

 }

 }

 //public void StartReadConnection()

 //{

 // pipeStreamRead.Connect();

 //}

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EndConnection\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will close both of the pipes that are connecting to the maze

 // generator from the server side. This should be called before the server program

 // ends in order to release the resources that were used.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void EndConnections()

 {

 //Close the pipe connection on the Server Side

 if (pipeStreamRead.IsConnected)

 {

 pipeStreamRead.Close();

 }

 if (pipeStreamWrite.IsConnected)

 {

 pipeStreamWrite.WaitForPipeDrain();

 pipeStreamWrite.Close();

 }

 }

 //Movement Functions

 //Each of these functions will send a code through the

 //pipe in order to inform the maze what direction to

 //move or where to place the cursor.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveSetLocation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will place the cursor in a set place in the maze. For this it

 // will require two parameters both of which are integers. The first will be used

 // for the row and the second will be for the column. The function will then send

 // the code for the move to the client and then will send the row followed by the

 // column. Nothing will be returned.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveSetLocation(int row, int column)

 {

 string temp = "moveLoc" + "," + row.ToString() + "," + column.ToString();

 SendMessage(temp);

 //string loc = row.ToString() + "," + column.ToString();

 //SendMessage(loc);

 //SendMessage(row.ToString());

 //SendMessage(column.ToString());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the client the code for Moving up one place in

 // the maze. Nothing will be returned.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveUp()

 {

 //Send the move up code (1)

 string temp = "up";

 SendMessage(temp);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the client the code for Moving down one place in

 // the maze. Nothing will be returned.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveDown()

 {

 string temp = "down";

 SendMessage(temp);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the client the code for Moving right one place in

 // the maze. Nothing will be returned.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveRight()

 {

 string temp = "right";

 SendMessage(temp);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the client the code for Moving left one place in

 // the maze. Nothing will be returned.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveLeft()

 {

 string temp = "left";

 SendMessage(temp);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWall\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will return an int based on whether there

 // is a wall in the specified direction within one step.

 // The function will return a -1 if there is an error or

 // the movement was out of the maze, a 0 if it was a wall,

 // or a 1 if it was a path.

 // For example if the path is a single line then this

 // function will return a 0 for the walls on the sides

 // of the cursor while the open paths in front and behind

 // will return a 1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWall(int direction)

 {

 string temp = "wallCheck";

 SendMessage(temp);

 SendMessage(direction.ToString()); //Must send which direction to check

 //Obtain the rersult

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Up from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallUp()

 {

 string temp = "checkWallUp";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Down from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallDown()

 {

 string temp = "checkWallDown";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Left from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallLeft()

 {

 string temp = "checkWallLeft";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Right from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallRight()

 {

 string temp = "checkWallRight";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*GetCurRow\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the code for retrieving the current

 // row to the client which will then return the row number of the

 // current position in the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int GetCurRow()

 {

 string temp = "curRow";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*GetCurCol\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will send the code for retrieving the

 // current column to the client which will then return the

 // column number of the current position in the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int GetCurCol()

 {

 string temp = "curCol";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckEnd\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if the end of he maze

 // has been reached or not. It will return a 1 if it has

 // or a 0 if it has not.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckEnd()

 {

 string temp = "endCheck";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit up from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathUp()

 {

 string temp = "checkUp";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit down from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathDown()

 {

 string temp = "checkDown";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit Left from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathLeft()

 {

 string temp = "checkLeft";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit Right from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathRight()

 {

 string temp = "checkRight";

 SendMessage(temp);

 return Convert.ToInt32(ReadMessage());

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SendMessage\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function is used to send messages to the client program. However

 // it will only be called by the functions and will not be called by the

 // user.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void SendMessage(string message)

 {

 StreamWriter sw = new StreamWriter(pipeStreamWrite);

 sw.AutoFlush = true;

 sw.WriteLine(message);

 }//End of SendMessage

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ReadMessage\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will retrieve messages from the client. However it will

 // not be called by the user and will only be used by the functions

 // themselves.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private string ReadMessage()

 {

 StreamReader sr = new StreamReader(pipeStreamRead);

 return sr.ReadLine();

 }//End ReadMessage

 }

}

**Maze Recursion Example:**

#include <iostream>

#include <string>

using namespace std;

//Connect to the Library of functions

#import "..\MazeFunctionLibrary\bin\Release\MazeFunctionLibrary.tlb" raw\_interfaces\_only

using namespace MazeFunctionLibrary;

bool MazeRecursion(long curRow, long curCol);

void waitFunction();

bool found = false;

long \* result = new long;

long \* result2 = new long;

long \* result3 = new long;

//Create the object for the rmgLibrary functions

HRESULT hr = CoInitialize(NULL);

rmgLibraryPtr rmgPtr(\_\_uuidof(ManagedClass));

int main()

{

 string temp;

 do

 {

 cout << "We will start when you type go." << endl;

 cin >> temp;

 }

 while(temp != "go");

 //Start the Connection

 rmgPtr->StartConnection();

 //Get the current row and column needed to start the recursion

 rmgPtr->GetCurRow(result);

 rmgPtr->GetCurCol(result2);

 //Call the Recursive function

 if(MazeRecursion(\*result, \*result2) == true)

 {

 cout << "FOUND THE END" << endl;

 }

 else

 {

 cout << "WHAT!!!!!" << endl;

 }

 cout << "Press a button and press enter to end" << endl;

 cin >> temp;

 //Close the connections between the client and server

 rmgPtr->EndConnections();

 return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MazeRecursion\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This function will use recursion in order to ry and reach the end of the

// maze. It will move through a set order, it will first move down then left,

// right, and then up. It will check and see if there is a wall in the direcion

// before moving the cursor. If the square is a wall then it will not move. The

// function will call MazeRecursion after each move in order to check the

// directions for the square that has been moved to. The function will return

// true if the endinf was found or false if it was not.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool MazeRecursion(long curRow, long curCol)

{

 if(found == true)

 {

 return true;

 }

 else

 {

 rmgPtr->CheckEnd(result);

 if(\*result == 1) //Found Finish Line

 {

 found = true;

 return true;

 }

 }

 //Check down

 //1s = been path

 //2s = open path

 rmgPtr->CheckPathDown(result);

 if(\*result == 2) //Path

 {

 rmgPtr->MoveDown();

 //Save new current spot

 rmgPtr->GetCurRow(result2);

 rmgPtr->GetCurCol(result3);

 /\*temp = curXLoc;

 temp2 = curYLoc;

 mazeRecursion(curXLoc, curYLoc);\*/

 waitFunction();

 MazeRecursion(\*result2,\*result3);

 }

 //Then return to old position

 rmgPtr->MoveSetLocation(curRow,curCol);

 rmgPtr->CheckPathLeft(result);

 if(\*result == 2)

 {

 rmgPtr->MoveLeft();

 rmgPtr->GetCurRow(result2);

 rmgPtr->GetCurCol(result3);

 waitFunction();

 MazeRecursion(\*result2,\*result3);

 }

 rmgPtr->MoveSetLocation(curRow,curCol);

 rmgPtr->CheckPathRight(result);

 if(\*result == 2)

 {

 rmgPtr->MoveRight();

 rmgPtr->GetCurRow(result2);

 rmgPtr->GetCurCol(result3);

 waitFunction();

 MazeRecursion(\*result2,\*result3);

 }

 rmgPtr->MoveSetLocation(curRow,curCol);

 rmgPtr->CheckPathUp(result);

 if(\*result == 2)

 {

 rmgPtr->MoveUp();

 rmgPtr->GetCurRow(result2);

 rmgPtr->GetCurCol(result3);

 waitFunction();

 MazeRecursion(\*result2,\*result3);

 }

 if(found == true)

 {

 return true;

 }

 //rmgPtr->CheckPathDown(result);

 //if(\*result == 1) //Path

 //{

 // rmgPtr->MoveDown();

 // MazeRecursion();

 //}

 //rmgPtr->CheckPathLeft(result);

 //if(\*result == 1)

 //{

 // rmgPtr->MoveLeft();

 // MazeRecursion();

 //}

 //rmgPtr->CheckPathRight(result);

 //if(\*result == 1)

 //{

 // rmgPtr->MoveRight();

 // MazeRecursion();

 //}

 //rmgPtr->CheckPathUp(result);

 //if(\*result == 1)

 //{

 // rmgPtr->MoveUp();

 // MazeRecursion();

 //}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WaitFunction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This function just runs a while loop that counts to a set number

// and is used to slow the recursive program down.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void waitFunction()

{

 int i = 0;

 while(i < 10500000)

 {

 i++;

 }

}

**Recursive Maze Generator:**

 **MainForm:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.IO.Pipes;

using System.Threading;

//Close the pipeStreamWrite when the form closes

//Open the pipeStreamWrite when the form is created

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Need to create\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- Move tracker so that user can see past moves and track the recursion

- Create Status Bar at bottom to show current location in x and y coord

- Also do the New, Save, and Save As features

- The Undo feature can also be implemented somewhere

- Finish implementing the help menu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Notes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- Might simply create a function that writes the string parameter to

 a file for the server to read and then reuse it when needed since they

 will all be written to the same file.

- ALL files will be written to and read from the RecursiveMazeMain folder

 area since it should not matter where the user creates their folder or

 what they name it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace RecursiveMazeMain

{

 public partial class RecursiveMazeMainForm : Form

 {

 //Need items for the maze squares such as color

 private Color WALL = Color.Gray; //0 in File

 private Color CURRENT = Color.Gold; //Not in File

 private Color ERRORSPACE = Color.Red; //Not in File

 private Color BEENTO = Color.Green; //Not in File

 private Color PASSABLE = Color.LightGray; //1 in File

 private Color START = Color.Yellow; //5 in File

 private Color FINISH = Color.RoyalBlue; //6 in File

 private Color FINISHCURRENT = Color.Blue; //Not in File

 private Color ERRORCURRENT = Color.Orange; //7 in File

 //Constants

 private const int SQUARESIZE = 50; //Height and Width of maze squares

 private const int STARTLOCATIONX = 300; //x coordinate to start maze

 private const int STARTLOCATIONY = 0; //y coordinate to start maze

 //This will be used to receive data from the server

 private const string PIPEREADING = "rmgPipeServer";

 NamedPipeClientStream pipeStreamRead = new NamedPipeClientStream(PIPEREADING);

 //StreamWriter sw = new StreamWriter(pipeStreamWrite);

 //This will be used to send data to the Server

 private const string PIPESENDING = "rmgPipeClient";

 System.IO.Pipes.NamedPipeServerStream pipeStreamWrite = new NamedPipeServerStream(PIPESENDING);

 //Create the thread

 private Thread pipeThread;

 private int sleepTime = 100;

 private bool threadOn = false;

 private MazeDoc mazeData;

 private ListBox moveLBox;

 public RecursiveMazeMainForm()

 {

 InitializeComponent();

 mazeData = new MazeDoc();

 //this.mazeData.Open += new System.EventHandler(OpenDocument);

 this.mazeData.UpdateMoveLog += new System.EventHandler(UpdateMoveLog);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OpenMenuClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // When the open menu item is clicked it will display the open

 // dialog in which the user can then open a maze that has already

 // been created. This will only open files with the extension rmg.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void openToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 bool openCheck = mazeData.Open();

 if (openCheck)

 {

 //Hide intro label and display the maze

 if (IntroLabel.Visible)

 {

 IntroLabel.Hide();

 }

 if (MazeRowTextBox.Visible == false)

 {

 MazeRowTextBox.Show();

 MazeColumnTextBox.Show();

 MoveMazeCursorButton.Show();

 RowLabel.Show();

 ColumnLabel.Show();

 StartConnectionButton.Show();

 ManualMovementButton.Show();

 ThreadControlCheckBox.Show();

 //finish creating and showing the list box

 moveLBox = new ListBox();

 moveLBox.Location = new Point(65, 50);

 moveLBox.Width = 150;

 moveLBox.Height = 250;

 Controls.Add(moveLBox);

 Label lBoxLabel = new Label();

 lBoxLabel.Width = 280;

 lBoxLabel.Text = "Move List (row number, column number)";

 lBoxLabel.Location = new Point(65, 35);

 Controls.Add(lBoxLabel);

 //Display the row and column labels

 Label rowInfoLabel = new Label();

 rowInfoLabel.Width = 55;

 rowInfoLabel.Text = "Row 0:";

 rowInfoLabel.Location = new Point(290, 70);

 Controls.Add(rowInfoLabel);

 Label columnInfoLabel = new Label();

 columnInfoLabel.Width = 55;

 columnInfoLabel.Text = "Column 0:";

 columnInfoLabel.Location = new Point(345, 27);

 Controls.Add(columnInfoLabel);

 }

 Invalidate();

 }

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*newToolStripMenuItem\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will display the new maze creation forms and then save the

 // newly created maze if the user desires. The save is done by calling the save

 // function from mazeDoc. This function will also create new instances of the

 // NewMazeStartForm and the NewMazeForm and will then display them.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void newToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 //Call the new maze form which will enable the user to create

 //a new maze either through randomness or by user controls

 NewMazeStartForm NMForm = new NewMazeStartForm();

 NMForm.ShowDialog();

 if (NMForm.DialogResult == DialogResult.OK)

 {

 int newRow = NMForm.NewRows;

 int newCol = NMForm.NewColumns;

 //MessageBox.Show(newRow.ToString());

 NewMazeForm NMForm2 = new NewMazeForm();

 NMForm2.NewMazeRow = newRow;

 NMForm2.NewMazeColumn = newCol;

 NMForm2.StartButtons();

 MessageBox.Show(NMForm2.NewMazeRow.ToString());

 NMForm2.ShowDialog();

 if(NMForm2.DialogResult == DialogResult.OK)

 {

 //obtain the data

 int[,] tempMaze = new int[NMForm2.NewMazeRow, NMForm2.NewMazeColumn];

 tempMaze = NMForm2.ObtainNewMaze();

 mazeData.Save(tempMaze, NMForm2.NewMazeRow, NMForm2.NewMazeColumn);

 }

 }

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HowToPlayMenuClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be called whenever the how to play menu item is clicked.

 // It will then create a new instance of the help form and then display it.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void HowToPlayToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 MazeHelpForm mazeHelp = new MazeHelpForm();

 mazeHelp.Show();

 }

 //This function is mostly used to test the creation of a solid

 //colored rectangle and testing the colors

 private void FillRectangleInt(object sender, EventArgs e)

 {

 // Create solid brush.

 SolidBrush blueBrush = new SolidBrush(Color.Blue);

 SolidBrush testing = new SolidBrush(Color.LightGray);

 Graphics t1 = this.CreateGraphics();

 // Create location and size of rectangle.

 int x = 350;

 int y = 50;

 int width = 50;

 int height = 50;

 IntroLabel.Hide();

 t1.FillRectangle(testing, x, y, width, height);

 testing.Color = Color.Gray;

 t1.FillRectangle(testing, x + width, y, width, height);

 // Fill rectangle to screen.

 //e.Graphics.FillRectangle(blueBrush, x, y, width, height);

 }//End of FillRectangleInt

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*mazePrintScreen\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called when paint event for the form is called. It will

 // display the maze if it has been opened and created in the mazDoc and will

 // update the maze when changes are made. It will also create a thread that will

 // check and see if the program has recieved any information from the server.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void mazePrintScreen(object sender, PaintEventArgs e)

 {

 if (mazeData.MazeCreated) //equals true

 {

 if (ThreadControlCheckBox.Checked == true)

 {

 if (ManualMovementButton.Enabled != false)

 {

 ManualMovementButton.Enabled = false;

 }

 pipeThread = new Thread(new ThreadStart(PipeThreadWork));

 pipeThread.Start();

 }

 else

 {

 //pipeThread.Abort();

 ManualMovementButton.Enabled = true;

 }

 //first create the graphics and brush in order to create the maze with

 Graphics mazeGraphics = this.CreateGraphics();

 SolidBrush mazeColoror = new SolidBrush(Color.Gray);

 //paint the maze on the screen

 for (int i = 0; i < mazeData.Rows; i++) //In charge of the rows

 {

 for (int j = 0; j < mazeData.Columns; j++) //In charge of the columns

 {

 //Obtain the color

 switch (mazeData.MazeBoard[i, j])

 {

 case 0:

 //Wall

 mazeColoror.Color = Color.Gray;

 break;

 case 1:

 mazeColoror.Color = Color.LightGray;

 break;

 case 2:

 mazeColoror.Color = Color.Green;

 break;

 case 3:

 mazeColoror.Color = Color.Red;

 break;

 case 4:

 mazeColoror.Color = Color.Gold;

 break;

 case 5:

 mazeColoror.Color = Color.Yellow;

 break;

 case 6:

 mazeColoror.Color = Color.RoyalBlue;

 break;

 case 7:

 mazeColoror.Color = Color.Orange;

 break;

 case 8:

 mazeColoror.Color = Color.Blue;

 break;

 default:

 mazeColoror.Color = Color.LightGray;

 break;

 }//end of Swich case

 //Draw the maze section

 mazeGraphics.FillRectangle(mazeColoror, STARTLOCATIONX + (SQUARESIZE \* (j + 1)), STARTLOCATIONY + (SQUARESIZE \* (i + 1)), SQUARESIZE, SQUARESIZE);

 }

 }

 }

 //else don't

 }//End of mazePrintScreen

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*moveMazeCursorButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will run whenever the move cursor button is clicked. It will first

 // make sure that the row and column text boxes are not empty. If they are it

 // will notify the user and will not change anything in the maze. If there are

 // values then it will call the mazeDoc function for moving to a set location.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MoveMazeCursorButton\_Click(object sender, EventArgs e)

 {

 //Make sure that there are enteries in the text boxes

 bool Test = false;

 try

 {

 if (MazeRowTextBox.Text.Trim() != "")

 {

 //Then not empty

 if (MazeColumnTextBox.Text.Trim() != "")

 {

 //Not Empty

 Test = true;

 }

 else

 {

 MazeColumnTextBox.Focus();

 throw new Exception();

 }

 }

 else

 {

 //Empty

 MazeRowTextBox.Focus();

 throw new Exception();

 }

 }

 catch (Exception ex)

 {

 MessageBox.Show("Please make sure that the row and column are entered.");

 }

 //move to coordinates on the maze

 if (Test == true)

 {

 int xCoord = Convert.ToInt32(MazeRowTextBox.Text);

 int yCoord = Convert.ToInt32(MazeColumnTextBox.Text);

 if (xCoord < 0 || xCoord > mazeData.Rows || yCoord < 0 || yCoord > mazeData.Columns)

 {

 //Then Error so display error window

 MessageBox.Show("Please keep the row and column in the maze.");

 }

 else

 {

 //The location is in the maze

 //So Mark the Spot you where at and color current location

 bool checker = mazeData.MoveLocation(xCoord, yCoord);

 if(checker == false)

 {

 MessageBox.Show("ERROR IN MOVE");

 }

 Invalidate();

 }

 }

 }//End of moveCursorLocationClick

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StartConnectionClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will run when the start connection button is clicked. It will

 // try and connect to the server by calling the start connections function

 // it will then disable the button.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void StartConnectionClick(object sender, EventArgs e)

 {

 //Will Start the connections and the thread that will

 //keep track of the pipes

 StartConnections();

 StartConnectionButton.Enabled = false;

 }//End start connection Button Click

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*aboutToolMenuItemClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will run when the about menu item is clicked. It will display

 // the about form to the user which holds basic information about the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void aboutToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 //Display the About Form

 AboutForm aboutForm = new AboutForm();

 aboutForm.Show();

 }//End of About Menu Item Click

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UpdateMoveLogEvent\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be in charge of updating the move log list box.

 // To do this it will be called from the mazeDoc which is in charge

 // of actually assigning the movement through the maze. This will

 // store the movements in the movement list box.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void UpdateMoveLog(object sender, EventArgs e)

 {

 moveLBox.Items.Add(mazeData.MoveData);

 }//End of UpdateMoveLog

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SendData\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be called when ever something needs to be

 // send over to the Server (other program). It will take a string

 // that will contain the message as a parameter and return nothing.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void SendData(string message)

 {

 //string temp = "";

 StreamWriter sw = new StreamWriter(pipeStreamWrite);

 sw.AutoFlush = true;

 if (message.Trim() != "")

 {

 //okay to send

 sw.WriteLine(message);

 }

 else

 {

 //no message to send

 sw.WriteLine("Sorry no message was sent");

 }

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ReadData\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be used to read the data that is sent over

 // from the server (other program). It will not take any parameters

 // however it will return a string that contains the message that

 // was sent over.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private string ReadData()

 {

 string temp = "";

 StreamReader sr = new StreamReader(pipeStreamRead);

 if ((temp = sr.ReadLine()) != null)

 {

 return temp;

 }

 else

 {

 return "";

 }

 //return temp;

 }//End of ReadData

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StartConnections\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be used to set up the connections between the

 // two programs.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void StartConnections()

 {

 //Start the thread

 //pipeThread = new Thread(new ThreadStart(PipeThreadWork));

 //pipeThread.Start();

 //Connect the read connection

 pipeStreamRead.Connect(120000); //Will try for up to two minutes

 //MessageBox.Show("READ CONNECTED");

 pipeStreamWrite.WaitForConnection(); //Start the Write server

 //MessageBox.Show("WRITE CONNECTED");

 }//End of StartConnections

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CornerClose\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called when the red X in the top right corer is

 // to close the program.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void CornerClose(object sender, FormClosingEventArgs e)

 {

 if (pipeStreamRead.IsConnected)

 {

 pipeStreamRead.Close();

 }

 if (pipeStreamWrite.IsConnected)

 {

 pipeStreamWrite.WaitForPipeDrain();

 pipeStreamWrite.Close();

 }

 //if (pipeThread.IsAlive)

 //{

 // pipeThread.Abort();

 //}

 //Create saves for new mazes

 }//End of Corner Close

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PipeThreadWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be called in order to connect to the pipes and retrieve

 // or send data. It will be the main work function for the thread.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 void PipeThreadWork()

 {

 //Call the specified function as read from file

 string orderNum = ""; //Might change to a string and split based on , for the last two functions

 string temp = "";

 //int goAhead = -1;

 //possible connect here

 //pipeStreamRead.Connect();

 //then read from the server

 //if (pipeStreamRead. != 0)

 //{

 if (pipeStreamRead.IsConnected)

 {

 orderNum = ReadData();

 // goAhead = 1;

 //}

 //else

 //{

 // goAhead = -1;

 // orderNum = "NODATA";

 //}

 //Make generic function for the sending of the data

 switch (orderNum)

 {

 case "0":

 break;

 case "up":

 //Move Up

 mazeData.MoveUp();

 break;

 case "down":

 //Move Down

 mazeData.MoveDown();

 break;

 case "left":

 //Move Left

 mazeData.MoveLeft();

 break;

 case "right":

 //Move Right

 mazeData.MoveRight();

 break;

 case "moveLoc":

 //Move Location

 //Must read the code then the coordinates

 string[] item = ReadData().Split(',');

 //int item = Convert.ToInt32(ReadData());

 //int item2 = Convert.ToInt32(ReadData());

 mazeData.MoveLocation(Convert.ToInt32(item[1]) ,Convert.ToInt32(item[2]));

 //mazeData.MoveLocation(

 break;

 case "wallCheckNO":

 //Check Wall

 //MessageBox.Show("HEHREHRHEHREHR");

 //Must read the code and then the direction

 temp = ReadData();

 MessageBox.Show(temp);

 SendData((mazeData.CheckWall(Convert.ToInt32(temp)).ToString())); //get the maze info and return

 //mazeData.CheckWall(

 break;

 case "checkUp":

 SendData(mazeData.CheckPathUp().ToString());

 break;

 case "checkDown":

 SendData(mazeData.CheckPathDown().ToString());

 break;

 case "checkLeft":

 SendData(mazeData.CheckPathLeft().ToString());

 break;

 case "checkRight":

 SendData(mazeData.CheckPathRight().ToString());

 break;

 case "curRow":

 //Return the current row

 SendData(mazeData.CurXLoc.ToString());

 break;

 case "curCol":

 //Return the current column

 SendData(mazeData.CurYLoc.ToString());

 break;

 case "endCheck":

 //Return 1 if end or 0 if not

 SendData(mazeData.CheckEnd().ToString());

 break;

 default:

 //No a valid code

 //Send over message saying so

 string []checkTwice = orderNum.Split(',');

 if (checkTwice[0] == "moveLoc")

 {

 mazeData.MoveLocation(Convert.ToInt32(checkTwice[1]), Convert.ToInt32(checkTwice[2]));

 }

 else

 {

 SendData("Invalid Code: Please try again.");

 }

 break;

 }

 // if (goAhead != -1)

 //{

 Invalidate();

 //}

 }

 Thread.Sleep(sleepTime);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ManualMoveClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called whenever the Manual Movement Button is clicked.

 // It will move the cursor through the maze one step at a time depending

 // of the directions recieved from the information sent from the server.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void ManualMove\_Click(object sender, EventArgs e)

 {

 //Call the specified function as read from file

 string orderNum = ""; //Might change to a string and split based on , for the last two functions

 string temp = "";

 int goAhead = -1;

 //possible connect here

 //pipeStreamRead.Connect();

 //then read from the server

 //if (pipeStreamRead. != 0)

 //{

 orderNum = ReadData();

 goAhead = 1;

 //}

 //else

 //{

 // goAhead = -1;

 // orderNum = "NODATA";

 //}

 //Make generic function for the sending of the data

 //switch (orderNum)

 //{

 // case "0":

 // break;

 // case "up":

 // //Move Up

 // mazeData.MoveUp();

 // break;

 // case "down":

 // //Move Down

 // mazeData.MoveDown();

 // break;

 // case "left":

 // //Move Left

 // mazeData.MoveLeft();

 // break;

 // case "right":

 // //Move Right

 // mazeData.MoveRight();

 // break;

 // case "moveLoc":

 // //Move Location

 // //Must read the code then the coordinates

 // mazeData.MoveLocation(Convert.ToInt32(ReadData()), Convert.ToInt32(ReadData()));

 // //mazeData.MoveLocation(

 // break;

 // case "wallCheck":

 // //Check Wall

 // //Must read the code and then the direction

 // temp = ReadData();

 // MessageBox.Show(temp);

 // int tempInt = mazeData.CheckWall(Convert.ToInt32(temp));

 // MessageBox.Show(tempInt.ToString());

 // SendData(tempInt.ToString()); //get the maze info and return

 // //mazeData.CheckWall(

 // break;

 // case "curRow":

 // //Return the current row

 // SendData(mazeData.CurXLoc.ToString());

 // break;

 // case "curCol":

 // //Return the current column

 // SendData(mazeData.CurYLoc.ToString());

 // break;

 // case "endCheck":

 // //Return 1 if end or 0 if not

 // SendData(mazeData.CheckEnd().ToString());

 // break;

 // default:

 // //No a valid code

 // //Send over message saying so

 // SendData("Invalid Code: Please try again.");

 // break;

 //}

 // Invalidate();

 switch (orderNum)

 {

 case "0":

 break;

 case "up":

 //Move Up

 mazeData.MoveUp();

 break;

 case "down":

 //Move Down

 mazeData.MoveDown();

 break;

 case "left":

 //Move Left

 mazeData.MoveLeft();

 break;

 case "right":

 //Move Right

 mazeData.MoveRight();

 break;

 case "moveLoc":

 //Move Location

 //Must read the code then the coordinates

 string[] item = ReadData().Split(',');

 //int item = Convert.ToInt32(ReadData());

 //int item2 = Convert.ToInt32(ReadData());

 mazeData.MoveLocation(Convert.ToInt32(item[1]), Convert.ToInt32(item[2]));

 //mazeData.MoveLocation(

 break;

 case "wallCheckNO":

 //Check Wall

 //MessageBox.Show("HEHREHRHEHREHR");

 //Must read the code and then the direction

 temp = ReadData();

 MessageBox.Show(temp);

 SendData((mazeData.CheckWall(Convert.ToInt32(temp)).ToString())); //get the maze info and return

 //mazeData.CheckWall(

 break;

 case "checkUp":

 SendData(mazeData.CheckPathUp().ToString());

 break;

 case "checkDown":

 SendData(mazeData.CheckPathDown().ToString());

 break;

 case "checkLeft":

 SendData(mazeData.CheckPathLeft().ToString());

 break;

 case "checkRight":

 SendData(mazeData.CheckPathRight().ToString());

 break;

 case "curRow":

 //Return the current row

 SendData(mazeData.CurXLoc.ToString());

 break;

 case "curCol":

 //Return the current column

 SendData(mazeData.CurYLoc.ToString());

 break;

 case "endCheck":

 //Return 1 if end or 0 if not

 SendData(mazeData.CheckEnd().ToString());

 break;

 default:

 //No a valid code

 //Send over message saying so

 string[] checkTwice = orderNum.Split(',');

 if (checkTwice[0] == "moveLoc")

 {

 mazeData.MoveLocation(Convert.ToInt32(checkTwice[1]), Convert.ToInt32(checkTwice[2]));

 }

 else

 {

 SendData("Invalid Code: Please try again.");

 }

 break;

 }

 // if (goAhead != -1)

 //{

 Invalidate();

 }//End of ManualMoveButtonClick

 }//End of Class

}

**MazeDoc:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using Microsoft.Win32;

namespace RecursiveMazeMain

{

 class MazeDoc

 {

 //Basic Document Data

 //Obtained from the MyDocumentClass in lab 6 the DocView lab

 private bool modified = false;

 private string fileName = "";

 private string defaultFileName = "Untitled";

 private string fileExtension = "rmg";

 private string fileExtensionDesc = "Recursive Maze Generator";

 private bool registerFileExtension = true;

 private string filter = "RMG Files (\*.rmg) |\*.rmg";

 //Specific Data

 private int rows;

 private int columns;

 private bool mazeCreated;

 private int[,] mazeBoard; //The maze setup data

 private int curXLoc;

 private int curYLoc;

 private int finishXLoc;

 private int finishYLoc;

 private string moveData; //Used to send back the movement to the move list box

 //Specific Constants

 //private const int MAXROWS = 10;

 //private const int MAXCOLUMNS = 10;

 private const int WALLCOLOR = 0; //Gray

 private const int PASSABLECOLOR = 1; //Light Gray to White

 private const int BEENPATHCOLOR = 2; //Green

 private const int ERRORCOLOR = 3; //Red

 private const int CURRENTLOCCOLOR = 4; //Gold

 private const int FINISHCOLOR = 6; //Royal Blue

 private const int FINISHCURRENTCOLOR = 8; //Blue

 private const int ERRORCURRENTLOCCOLOR = 7; //Orange

 //Will have server part read from a specified path

 private const string TEMPFILENAMEFROMMAZE = "tmpMazeToServerCom.txt";

 private const string TEMPFILENAMEFROMSERVER = "tempServerToMazeCom.txt";

 //Events

 public event EventHandler UpdateMoveLog;

 //Properties

 //Get and Set functions for the various data

 public int Rows

 {

 get

 {

 return rows;

 }

 }

 public int Columns

 {

 get

 {

 return columns;

 }

 }

 public bool MazeCreated

 {

 get

 {

 return mazeCreated;

 }

 }

 public int[,] MazeBoard

 {

 get

 {

 return mazeBoard;

 }

 }

 public int CurXLoc

 {

 get

 {

 return curXLoc;

 }

 set

 {

 curXLoc = value;

 }

 }

 public int CurYLoc

 {

 get

 {

 return curYLoc;

 }

 set

 {

 curYLoc = value;

 }

 }

 public string MoveData

 {

 get

 {

 return moveData;

 }

 }

 //Events

 public event EventHandler OpenDocument;

 public MazeDoc()

 {

 modified = false;

 mazeCreated = false; //To tell if maze has been created or loaded

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Open\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This funtion will be used by the open command in the main form

 // in order to call the open work function which will display the

 // open dialog which will be used to open existing mazes.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public bool Open()

 {

 return Open(null);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OpenWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will do all of the work to display the open

 // dialog. The Basic outline for this function is from the

 // CSCI 350 Lab 6 DocVeiw by Dr. Bonnie Mcvey.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private bool Open(string newFileName)

 {

 //open the open screen

 OpenFileDialog dlg = new System.Windows.Forms.OpenFileDialog();

 if (string.IsNullOrEmpty(newFileName))

 {

 // Set up dialog

 dlg.DefaultExt = fileExtension;

 dlg.Filter = filter;

 // Get the file to open

 DialogResult res = dlg.ShowDialog();

 if (res != DialogResult.OK)

 {

 return false;

 }

 newFileName = dlg.FileName;

 }

 try

 {

 //Obtain the data from the file

 using (StreamReader sr = new StreamReader(newFileName))

 {

 //read the number of rows and columns

 string rowCol = sr.ReadLine();

 rows = Convert.ToInt32(rowCol);

 rowCol = sr.ReadLine();

 columns = Convert.ToInt32(rowCol);

 //Retrieve the Maze Data

 //So the walls, paths, start, and finish

 //First Create a new instance of the maze board

 //with the obtained rows and columns data

 mazeBoard = new int[rows, columns];

 string mazeData; //Data read from file

 //obtain the values for each item in the maze

 for (int i = 0; i < rows; i++) //The number of times to get all rows

 {

 mazeData = sr.ReadLine(); //Get next line of data

 string[] mazeDataSplit = mazeData.Split(' ');

 for (int j = 0; j < columns; j++) //Number needed to get all columns

 {

 mazeBoard[i, j] = Convert.ToInt32(mazeDataSplit[j]);

 if(Convert.ToInt32(mazeDataSplit[j]) == 5)

 {

 curXLoc = i;

 curYLoc = j;

 }

 if (Convert.ToInt32(mazeDataSplit[j]) == 6)

 {

 finishXLoc = i;

 finishYLoc = j;

 }

 }

 }

 }

 }

 catch (Exception e)

 {

 MessageBox.Show("Can't open file: " + newFileName + "\r\n" + e.Message, Application.ProductName);

 return false;

 }

 //More will follow

 // Success

 if (this.OpenDocument != null)

 this.OpenDocument(this, EventArgs.Empty);

 mazeCreated = true;

 return true;

 }//End of OpenWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Save\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the SaveWork function. It will

 // recieve a 2d int array which will hold the information that

 // hold the maze data and an int for a row and a column. It will

 // then send the information to the work function.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public bool Save(int[,] tempMaze, int row, int col)

 {

 return SaveWork(tempMaze, rows, col);

 }//End of Save

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SaveWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will save the new maze to a file. It will recieve

 // the maze data via a 2d array and the row and column data via

 // integers. It will then open the save dialog which will allow

 // the user to decide where to save the maze and what it will be

 // called. The function will fist save the row and column to the

 // maze and will then save the data for what is in the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private bool SaveWork(int[,] tempMaze, int row, int col)

 {

 string newFileName = fileName;

 SaveFileDialog sdlg = new SaveFileDialog();

 sdlg.DefaultExt = fileExtension;

 sdlg.Filter = filter;

 if(!string.IsNullOrEmpty(newFileName))

 {

 sdlg.InitialDirectory = Path.GetDirectoryName(newFileName);

 sdlg.FileName = Path.GetFileName(newFileName);

 }

 else

 {

 sdlg.FileName = this.fileName;

 }

 DialogResult res = sdlg.ShowDialog();

 if (res != DialogResult.OK) return false;

 newFileName = sdlg.FileName;

 try

 {

 using (StreamWriter sw = new StreamWriter(newFileName))

 {

 int r = tempMaze.GetUpperBound(0) + 1;

 int c = tempMaze.GetUpperBound(tempMaze.Rank - 1) + 1;

 sw.WriteLine(r.ToString());

 sw.WriteLine(c.ToString());

 //enter the maze data

 string tempString = "";

 //MessageBox.Show(r.ToString());

 for (int i = 0; i < r; i++)

 {

 for (int j = 0; j < c; j++)

 {

 tempString += tempMaze[i, j].ToString() + " ";

 }

 sw.WriteLine(tempString);

 tempString = "";

 }

 //sw.WriteLine(rows.ToString() + " " + columns.ToString());

 ////enter the number of moves that were used

 //sw.WriteLine(moveCount);

 ////enter the move list which is stored in as strings

 ////sw.WriteLine(dataString);

 //for (int i = 0; i < moveCount; i++)

 //{

 // sw.WriteLine(dataString[i]);

 //}

 //sw.WriteLine(ptCount.ToString());

 //for (int i = 0; i < ptCount; i++)

 // sw.WriteLine(myPoints[i].X.ToString() + " "

 // + myPoints[i].Y.ToString());

 }

 }

 catch (Exception e)

 {

 MessageBox.Show("Can't save file: " + newFileName + "\r\n" + e.Message, Application.ProductName);

 return false;

 }

 return true;

 }//End SaveWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveLocation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This is the driver function for the MoveLocationWork

 // function which will move the current position cursor to

 // a set location.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public bool MoveLocation(int x, int y)

 {

 return MoveLocationWork(x,y);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveLocationWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This is the work function for Move Location. It will move

 // the current position marker to a specified location in the

 // maze. It will allow movement onto walls but not outside the

 // boundary of the maze. So it will not be able to move to (-1,3).

 // This function will return a bool in order to make sure that the

 // coordinates are within the correct boundary of the maze. If so

 // then true is returned. Else False.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private bool MoveLocationWork(int x, int y)

 {

 //Check and make sure that the location is in bounds.

 //(This should have been done at the maze but will check again)

 if (x >= 0 && x < rows && y >= 0 && y < columns)

 {

 moveData = "From: (" + curXLoc.ToString() + ", " + curYLoc.ToString() + ") To: (" + x.ToString() + ", " + y.ToString() + ") ";

 //Good Data so set the values

 if (mazeBoard[curXLoc, curYLoc] == FINISHCURRENTCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = FINISHCOLOR;

 }

 else if (mazeBoard[curXLoc, curYLoc] != ERRORCURRENTLOCCOLOR && mazeBoard[curXLoc, curYLoc] != WALLCOLOR)

 {

 //Not a wall

 mazeBoard[curXLoc, curYLoc] = BEENPATHCOLOR;

 }

 else

 {

 mazeBoard[curXLoc, curYLoc] = ERRORCOLOR;

 }

 if (mazeBoard[x, y] == WALLCOLOR || mazeBoard[x, y] == ERRORCOLOR)

 {

 //Heading into a wall

 moveData += "Wall";

 mazeBoard[x, y] = ERRORCURRENTLOCCOLOR;

 }

 else if (mazeBoard[x, y] == FINISHCOLOR)

 {

 moveData += "Finished";

 mazeBoard[x, y] = FINISHCURRENTCOLOR;

 }

 else

 {

 //Passable area

 moveData += "Path";

 mazeBoard[x, y] = CURRENTLOCCOLOR;

 }

 curXLoc = x;

 curYLoc = y;

 if (UpdateMoveLog != null)

 {

 this.UpdateMoveLog(this, EventArgs.Empty);

 }

 return true;

 }

 else

 {

 return false;

 }

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the MoveUpWork function in order

 // to move the items in the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveUp()

 {

 MoveUpWork();

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveUpWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This is the worker of the MoveUp function. It will move the

 // current position marker in the maze up one unit. This will

 // allow movement through walls in order to show the steps

 // taken. However it will not allow the marker to move outside

 // the set boundary of the maze. So it will not move to (-1,3).

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MoveUpWork()

 {

 if (curXLoc - 1 < 0)

 {

 //Error Out of bounds

 MessageBox.Show("The current location would be outside of the maze boundary. Nothing was changed.");

 }

 else

 {

 //Is good

 if (mazeBoard[curXLoc, curYLoc] == ERRORCURRENTLOCCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = ERRORCOLOR;

 }

 else if (mazeBoard[curXLoc, curYLoc] == FINISHCURRENTCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = FINISHCOLOR;

 }

 else

 {

 mazeBoard[curXLoc, curYLoc] = BEENPATHCOLOR;

 }

 moveData = "From: (" + curXLoc.ToString() + ", " + curYLoc.ToString() + ") To: (" + (curXLoc - 1).ToString() + ", " + curYLoc.ToString() + ") ";

 if (mazeBoard[curXLoc - 1, curYLoc] == WALLCOLOR || mazeBoard[curXLoc - 1, curYLoc] == ERRORCOLOR)

 {

 //Then is a wall

 moveData += "Wall";

 mazeBoard[curXLoc - 1, curYLoc] = ERRORCURRENTLOCCOLOR;

 curXLoc--;

 }

 else if (mazeBoard[curXLoc - 1, curYLoc] == FINISHCOLOR || mazeBoard[curXLoc - 1, curYLoc] == FINISHCURRENTCOLOR)

 {

 //Then is the Finish Line

 moveData += "Finished";

 mazeBoard[curXLoc - 1, curYLoc] = FINISHCURRENTCOLOR;

 curXLoc--;

 }

 else

 {

 moveData += "Path";

 mazeBoard[curXLoc - 1, curYLoc] = CURRENTLOCCOLOR;

 curXLoc--;

 }

 if (UpdateMoveLog != null)

 {

 this.UpdateMoveLog(this, EventArgs.Empty);

 }

 }

 }//End of MoveUpWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be used to call the MoveDownWork

 // funcion in order to move the current position down

 // one unit.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveDown()

 {

 MoveDownWork();

 }//End of MoveDown

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveDownWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will provide the work for the move down

 // function. It will move the current position marker down

 // one unit. It will allow movement through wall but not

 // outside of the maze boundaries. So it will not allow

 // movement to (8,3) in a maze that is 7x3. It will also

 // update the movement log in the main form.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MoveDownWork()

 {

 if (curXLoc + 1 > rows-1)

 {

 //Error Out of bounds

 MessageBox.Show("The current location would be outside of the maze boundary. Nothing was changed.");

 }

 else

 {

 //Is good

 if (mazeBoard[curXLoc, curYLoc] == ERRORCURRENTLOCCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = ERRORCOLOR;

 }

 else if (mazeBoard[curXLoc, curYLoc] == FINISHCURRENTCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = FINISHCOLOR;

 }

 else

 {

 mazeBoard[curXLoc, curYLoc] = BEENPATHCOLOR;

 }

 moveData = "From: (" + curXLoc.ToString() + ", " + curYLoc.ToString() + ") To: (" + (curXLoc + 1).ToString() + ", " + curYLoc.ToString() + ") ";

 if (mazeBoard[curXLoc + 1, curYLoc] == WALLCOLOR || mazeBoard[curXLoc + 1, curYLoc] == ERRORCOLOR)

 {

 //Then is a wall

 moveData += "Wall";

 mazeBoard[curXLoc + 1, curYLoc] = ERRORCURRENTLOCCOLOR;

 curXLoc++;

 }

 else if (mazeBoard[curXLoc + 1, curYLoc] == FINISHCOLOR || mazeBoard[curXLoc + 1, curYLoc] == FINISHCURRENTCOLOR)

 {

 //Then is the Finish Line

 moveData += "Finished";

 mazeBoard[curXLoc + 1, curYLoc] = FINISHCURRENTCOLOR;

 curXLoc++;

 }

 else

 {

 moveData += "Path";

 mazeBoard[curXLoc + 1, curYLoc] = CURRENTLOCCOLOR;

 curXLoc++;

 }

 if (UpdateMoveLog != null)

 {

 this.UpdateMoveLog(this, EventArgs.Empty);

 }

 }

 }//End of MoveDownWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the move left work function in order

 // to move the current position marker left one unit.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveLeft()

 {

 MoveLeftWork();

 }//End of MoveLeft

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveLeftWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This will move the current position marker left one unit. It

 // will allow movement through walls but not outisde the boundary

 // of the maze. So it will not allow movement to (3,-1) since the

 // lowest value of the mazes columns is 0 no negative value would

 // be accepted.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MoveLeftWork()

 {

 if (curYLoc - 1 < 0)

 {

 //Error Out of bounds

 MessageBox.Show("The current location would be outside of the maze boundary. Nothing was changed.");

 }

 else

 {

 //Is good

 if (mazeBoard[curXLoc, curYLoc] == ERRORCURRENTLOCCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = ERRORCOLOR;

 }

 else if (mazeBoard[curXLoc, curYLoc] == FINISHCURRENTCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = FINISHCOLOR;

 }

 else

 {

 mazeBoard[curXLoc, curYLoc] = BEENPATHCOLOR;

 }

 moveData = "From: (" + curXLoc.ToString() + ", " + curYLoc.ToString() + ") To: (" + curXLoc.ToString() + ", " + (curYLoc - 1).ToString() + ") ";

 if (mazeBoard[curXLoc, curYLoc-1] == WALLCOLOR || mazeBoard[curXLoc, curYLoc-1] == ERRORCOLOR)

 {

 //Then is a wall

 moveData += "Wall";

 mazeBoard[curXLoc, curYLoc-1] = ERRORCURRENTLOCCOLOR;

 curYLoc--;

 }

 else if (mazeBoard[curXLoc, curYLoc - 1] == FINISHCOLOR || mazeBoard[curXLoc, curYLoc - 1] == FINISHCURRENTCOLOR)

 {

 //Then is the Finish Line

 moveData += "Finished";

 mazeBoard[curXLoc, curYLoc - 1] = FINISHCURRENTCOLOR;

 curYLoc--;

 }

 else

 {

 moveData += "Path";

 mazeBoard[curXLoc, curYLoc-1] = CURRENTLOCCOLOR;

 curYLoc--;

 }

 if (UpdateMoveLog != null)

 {

 this.UpdateMoveLog(this, EventArgs.Empty);

 }

 }

 }//End of MoveLeftWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This fucntion will call the MoveRightWork function in order

 // to move right one unit.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void MoveRight()

 {

 MoveRightWork();

 }//End of MoveRight

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MoveRightWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This will move the current position marker right one unit. It

 // will allow movement through walls but not outisde the boundary

 // of the maze. So it will not allow movement to (3,8) if the

 // max column for that maze was 7.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MoveRightWork()

 {

 if (curYLoc + 1 > columns-1)

 {

 //Error Out of bounds

 MessageBox.Show("The current location would be outside of the maze boundary. Nothing was changed.");

 }

 else

 {

 //Is good

 if (mazeBoard[curXLoc, curYLoc] == ERRORCURRENTLOCCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = ERRORCOLOR;

 }

 else if (mazeBoard[curXLoc, curYLoc] == FINISHCURRENTCOLOR)

 {

 mazeBoard[curXLoc, curYLoc] = FINISHCOLOR;

 }

 else

 {

 mazeBoard[curXLoc, curYLoc] = BEENPATHCOLOR;

 }

 moveData = "From: (" + curXLoc.ToString() + ", " + curYLoc.ToString() + ") To: (" + curXLoc.ToString() + ", " + (curYLoc + 1).ToString() + ") ";

 if (mazeBoard[curXLoc, curYLoc + 1] == WALLCOLOR || mazeBoard[curXLoc, curYLoc + 1] == ERRORCOLOR)

 {

 //Then is a wall

 moveData += "Wall";

 mazeBoard[curXLoc, curYLoc + 1] = ERRORCURRENTLOCCOLOR;

 curYLoc++;

 }

 else if (mazeBoard[curXLoc, curYLoc + 1] == FINISHCOLOR || mazeBoard[curXLoc, curYLoc + 1] == FINISHCURRENTCOLOR)

 {

 //Then is the Finish Line

 moveData += "Finished";

 mazeBoard[curXLoc, curYLoc + 1] = FINISHCURRENTCOLOR;

 curYLoc++;

 }

 else

 {

 moveData += "Path";

 mazeBoard[curXLoc, curYLoc + 1] = CURRENTLOCCOLOR;

 curYLoc++;

 }

 if (UpdateMoveLog != null)

 {

 this.UpdateMoveLog(this, EventArgs.Empty);

 }

 }

 }//End of MoveRightWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Up from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallUp()

 {

 int tempX = curXLoc;

 int tempY = curYLoc;

 tempX--;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else

 {

 if (mazeBoard[tempX, tempY] == WALLCOLOR)

 {

 return 0;

 }

 else

 {

 return 1;

 }

 }

 }//End of CheckWallUp

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // down from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallDown()

 {

 int tempX = curXLoc;

 int tempY = curYLoc;

 tempX++;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else

 {

 if (mazeBoard[tempX, tempY] == WALLCOLOR)

 {

 return 0;

 }

 else

 {

 return 1;

 }

 }

 }//End of CheckWallDown

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Left of the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallLeft()

 {

 int tempX = curXLoc;

 int tempY = curYLoc;

 tempY--;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else

 {

 if (mazeBoard[tempX, tempY] == WALLCOLOR)

 {

 return 0;

 }

 else

 {

 return 1;

 }

 }

 }//End of CheckWallLeft

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWallRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if there is a wall one unit

 // Right from the current position. If there is then it will return a

 // 0 if not it will return a 1. If the iem is outside of the maze

 // boundary it will return a -1.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWallRight()

 {

 int tempX = curXLoc;

 int tempY = curYLoc;

 tempY++;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else

 {

 if (mazeBoard[tempX, tempY] == WALLCOLOR)

 {

 return 0; //IS A WALL

 }

 else

 {

 return 1;

 }

 }

 }//End of CheckWallRight

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathUp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the CheckPathUpWork function in order

 // to check and see what type of square is one unit up from the

 // current position.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathUp()

 {

 return CheckPathUpWork();

 }//End of CheckPathUp

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathUpWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit up from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int CheckPathUpWork()

 {

 if (CheckWallUp() == 0)

 {

 return 0; //Means it is a wall

 }

 else

 {

 int tempX = curXLoc;

 tempX--;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (mazeBoard[tempX, curYLoc] == BEENPATHCOLOR)

 {

 return 1; //Been Path

 }

 else

 {

 return 2; //Not Yet Been

 }

 }

 }//End of CheckPathUpWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathDown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the CheckPathDownWork function in order

 // to check and see what type of square is one unit Down from the

 // current position.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathDown()

 {

 return CheckPathDownWork();

 }//End of CheckPathDown

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathDownWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit down from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int CheckPathDownWork()

 {

 if (CheckWallDown() == 0)

 {

 return 0; //Means it is a wall

 }

 else

 {

 int tempX = curXLoc;

 tempX++;

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (mazeBoard[tempX, curYLoc] == BEENPATHCOLOR)

 {

 return 1; //Been Path

 }

 else

 {

 return 2; //Not Yet Been

 }

 }

 }//End of CheckPathDownWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathLeft\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the CheckPathLeftWork function in order

 // to check and see what type of square is one unit Left from the

 // current position.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathLeft()

 {

 return CheckPathLeftWork();

 }//End of CheckPathLeft

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathLeftWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit left from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int CheckPathLeftWork()

 {

 if (CheckWallLeft() == 0)

 {

 return 0; //Means it is a wall

 }

 else

 {

 int tempY = curYLoc;

 tempY--;

 if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else if (mazeBoard[curXLoc, tempY] == BEENPATHCOLOR)

 {

 return 1; //Been Path

 }

 else

 {

 return 2; //Not Yet Been

 }

 }

 }//End of CheckPathLeftWork

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathRight\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the CheckPathRightWork function in order

 // to check and see what type of square is one unit Right from the

 // current position.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckPathRight()

 {

 return CheckPathRightWork();

 }//End of CheckPathRight

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckPathRightWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see what the value of the maze square

 // one unit right from the current position is. It will return a 0 if

 // it is a wall, a -1 if it is out of bounds, a 1 if it is marked as

 // having been to that square, or a 2 if the cursor has not yest moved

 // to that square.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int CheckPathRightWork()

 {

 if (CheckWallRight() == 0)

 {

 return 0; //Means it is a wall

 }

 else

 {

 int tempY = curYLoc;

 tempY++;

 if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else if (mazeBoard[curXLoc, tempY] == BEENPATHCOLOR)

 {

 return 1; //Been Path

 }

 else

 {

 return 2; //Not Yet Been

 }

 }

 }//End of CheckPathRightWork

 //Return -1 for outside maze or error, 0 for wall, 1 for path

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckWall\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function is supposed to be a generic way to check and see

 // if there is a wall in the direction that is sent as a parameter

 // however as it is currently it will not work.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckWall(int direction)

 {

 int tempX = curXLoc;

 int tempY = curYLoc;

 //if (direction == "up" || direction == "Up")

 //{

 // //if (mazeBoard[(curXLoc - 1), curYLoc] == WALLCOLOR)

 // //{

 // // return 0;

 // //}

 // //else

 // //{

 // // return 1;

 // //}

 //}

 switch (direction)

 {

 case 0: //Up

 tempX--;

 break;

 //if (mazeBoard[(curXLoc - 1), curYLoc] == WALLCOLOR)

 //{

 // return 0;

 //}

 //else

 //{

 // return 1;

 //}

 case 1: //Down

 tempX++;

 break;

 //if (mazeBoard[(curXLoc + 1), curYLoc] == WALLCOLOR)

 //{

 // return 0;

 //}

 //else

 //{

 // return 1;

 //}

 case 2: //Left

 tempY--;

 break;

 //if (mazeBoard[curXLoc, (curYLoc - 1)] == WALLCOLOR)

 //{

 // return 0;

 //}

 //else

 //{

 // return 1;

 //}

 case 3: //right

 tempY++;

 break;

 //if (mazeBoard[curXLoc, (curYLoc + 1)] == WALLCOLOR)

 //{

 // return 0;

 //}

 //else

 //{

 // return 1;

 //}

 default:

 MessageBox.Show("Invalid direction. Please enter up, down, left, or right");

 return -1;

 }

 if (tempX < 0 || tempX > rows - 1)

 {

 return -1;

 }

 else if (tempY < 0 || tempY > columns - 1)

 {

 return -1;

 }

 else

 {

 if (mazeBoard[tempX, tempY] == WALLCOLOR)

 {

 return 0;

 }

 else

 {

 return 1;

 }

 }

 }//End of Wall Check

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckEnd\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will call the CheckEndWork function in order to see

 // if the end of the maze has been reached or not.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int CheckEnd()

 {

 return CheckEndWork();

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CheckEndWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will check and see if the end of the maze has

 // been reached and will return a 1 if yes or a 0 if no.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int CheckEndWork()

 {

 if (curXLoc == finishXLoc && curYLoc == finishYLoc)

 {

 return 1;

 }

 else

 {

 return 0;

 }

 }

 }//End of MazeDoc Class

}

**NewMazeStartForm:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace RecursiveMazeMain

{

 public partial class NewMazeStartForm : Form

 {

 private int newRows;

 private int newColumns;

 public int NewRows

 {

 get

 {

 return Convert.ToInt32(NewRowComboBox.Text);

 }

 }

 public int NewColumns

 {

 get

 {

 return Convert.ToInt32(NewColumnComboBox.Text);

 }

 }

 public NewMazeStartForm()

 {

 InitializeComponent();

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OkButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called when the OK button is clicked and will

 // return a dialog result of OK.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void NewMazeFormOKButton\_Click(object sender, EventArgs e)

 {

 DialogResult = DialogResult.OK;

 }//End of NewMazeFormOKClick

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CancelButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called when the cancel button is clicked. It will

 // return a dialog result of cancel.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void NewMazeFormCancelButton\_Click(object sender, EventArgs e)

 {

 DialogResult = DialogResult.Cancel;

 }//End of NewMazeFormCancelClick

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ValidateRowData\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This validation event will be called whenever the row text box loses focus.

 // It will check and make sure that the row value is between 2 and 7.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void ValidateRowData(object sender, CancelEventArgs e)

 {

 if (Convert.ToInt32(NewRowComboBox.Text) < 2 || Convert.ToInt32(NewRowComboBox.Text) > 7)

 {

 MessageBox.Show("The rows must be beween 2 and 7.");

 NewRowComboBox.Focus();

 }

 else

 {

 newRows = Convert.ToInt32(NewRowComboBox.Text);

 }

 }//End of ValidateRowData

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ValidatecolumnData\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This validation event will be called whenever the row text box loses focus.

 // It will check and make sure that the column value is between 2 and 7.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void ValidateColumnData(object sender, CancelEventArgs e)

 {

 if (Convert.ToInt32(NewColumnComboBox.Text) < 2 || Convert.ToInt32(NewColumnComboBox.Text) > 7)

 {

 MessageBox.Show("The columns must be beween 2 and 7.");

 NewColumnComboBox.Focus();

 }

 else

 {

 newColumns = Convert.ToInt32(NewColumnComboBox.Text);

 }

 }//End of ValidateColumnData

 }

}

**NewMazeForm:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace RecursiveMazeMain

{

 public partial class NewMazeForm : Form

 {

 private int newMazeRow;

 private int newMazeColumn;

 private Button[,] newMazeBoard;

 private int[,] newMazeBoardValues;

 private int startAmount = 0;

 private int finishAmount = 0;

 private const int BUTTONSIZE = 50;

 private Color WALL = Color.Gray; //0

 private Color PATH = Color.LightGray; //1

 private Color START = Color.Yellow; //5

 private Color FINISH = Color.RoyalBlue; //6

 public int NewMazeRow

 {

 get

 {

 return newMazeRow;

 }

 set

 {

 newMazeRow = value;

 }

 }

 public int NewMazeColumn

 {

 get

 {

 return newMazeColumn;

 }

 set

 {

 newMazeColumn = value;

 }

 }

 //public int[,] NewMazeBoardValues

 //{

 // get

 // {

 // return NewMazeBoardValues;

 // }

 //}

 public NewMazeForm()

 {

 InitializeComponent();

 //InitializeButtons();

 //MessageBox.Show(newMazeColumn.ToString());

 //MessageBox.Show(newMazeRow.ToString());

 //MessageBox.Show("HERE");

 //Button test = new Button();

 //test.Size = new Size(BUTTONSIZE, BUTTONSIZE);

 //test.Location = new Point(100, 100);

 //this.Controls.Add(test);

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StartButtons\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function is used to call the Initialize Buttons function.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public void StartButtons()

 {

 InitializeButtons();

 }//End of StartButtons

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*InitialiazeButtons\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will display all of the buttons used to create the maze

 // by using the data that was obtained from the new maze start form.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void InitializeButtons()

 {

 //Create and display the buttons for the new maze

 if (newMazeBoard != null)

 {

 foreach (Button bn in newMazeBoard)

 {

 bn.Hide();

 }

 }

 newMazeBoard = new Button[newMazeRow, newMazeColumn];

 int r = newMazeBoard.GetUpperBound(0) + 1;

 int c = newMazeBoard.GetUpperBound(newMazeBoard.Rank - 1) + 1;

 int xloc = 50;

 int yloc = 50;

 for (int i = 0; i < r; i++)

 {

 for (int j = 0; j < c; j++)

 {

 newMazeBoard[i, j] = new Button();

 newMazeBoard[i, j].Size = new Size(BUTTONSIZE, BUTTONSIZE);

 newMazeBoard[i, j].Location = new Point(xloc, yloc);

 newMazeBoard[i, j].BackColor = WALL;

 //assign the click event

 newMazeBoard[i, j].Click += new EventHandler(MazeButtonClick);

 newMazeBoardValues = new int[newMazeRow, newMazeColumn];

 newMazeBoardValues[i, j] = 0;

 xloc += 50;

 }

 xloc = 50;

 yloc += 50;

 }

 //MessageBox.Show("HEREHERE");

 for (int i = 0; i < r; i++)

 {

 for (int j = 0; j < c; j++)

 {

 this.Controls.Add(newMazeBoard[i, j]);

 }

 }

 //Invalidate();

 }//End of Initialize Buttons

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ObtainNewMaze\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This function will be used to retrieve the maze from the new maze form

 // and will return the result of the worker function.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public int[,] ObtainNewMaze()

 {

 return ObtainNewMazeWork();

 }//End of ObtainNewMaze

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ObtainNewMazeWork\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This functon will provide then work for the function ObtainNewMaze. It will

 // return the values of the created maze based on the colors of the buttons

 // in the maze.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private int[,] ObtainNewMazeWork()

 {

 for (int i = 0; i < newMazeRow; i++)

 {

 for (int j = 0; j < newMazeColumn; j++)

 {

 if (newMazeBoard[i, j].BackColor == WALL)

 {

 newMazeBoardValues[i, j] = 0;

 }

 else if (newMazeBoard[i, j].BackColor == PATH)

 {

 newMazeBoardValues[i, j] = 1;

 }

 else if (newMazeBoard[i, j].BackColor == START)

 {

 newMazeBoardValues[i, j] = 5;

 }

 else if (newMazeBoard[i, j].BackColor == FINISH)

 {

 newMazeBoardValues[i, j] = 6;

 }

 }

 }

 return newMazeBoardValues;

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MazeButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will run when the maze buttons are clicked. It will change

 // the color of the buttons based on the amount of clicks. It will then

 // alter the amount of start and finish areas so that a maze can and must

 // have only one of each.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void MazeButtonClick(object sender, EventArgs e)

 {

 Button btn = (Button)sender;

 //bool temp;

 if (btn.BackColor == WALL)

 {

 btn.BackColor = PATH;

 }

 else if (btn.BackColor == PATH)

 {

 startAmount++;

 btn.BackColor = START;

 }

 else if (btn.BackColor == START)

 {

 if (startAmount > 0)

 {

 startAmount--;

 }

 finishAmount++;

 btn.BackColor = FINISH;

 }

 else

 {

 if (finishAmount > 0)

 {

 finishAmount--;

 }

 btn.BackColor = WALL;

 }

 }//End of button click

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SaveButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This will be called when the save button is clicked. It will first

 // make sure that there is only on start and finish in the maze. It will

 // then return an OK as the dialog result.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void SaveButton\_Click(object sender, EventArgs e)

 {

 if (startAmount == 1 && finishAmount == 1)

 {

 DialogResult = DialogResult.OK;

 }

 else

 {

 MessageBox.Show("Please make sure that there is one Start and Finish");

 MessageBox.Show(finishAmount.ToString());

 MessageBox.Show(startAmount.ToString());

 }

 }

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CancelButtonClick\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 // This event will be called when the cancel button is clicked. It will

 // return a dialog result of cancel.

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 private void CancelButton\_Click(object sender, EventArgs e)

 {

 DialogResult = DialogResult.Cancel;

 }

 }

}

**Test Programs:**

**RecursiveMazeTest:**

#include <iostream>

#include <string>

using namespace std;

//Include the library in the file

#import "..\CSCI 460\MazeFunctionLibrary\bin\Release\MazeFunctionLibrary.tlb" raw\_interfaces\_only

using namespace MazeFunctionLibrary;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Information\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This program uses a while loop in order to obtain the users directions which it then

// sends to the client maze program via the function library commands.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()

{

 //Create the object

 HRESULT hr = CoInitialize(NULL);

 rmgLibraryPtr rmgPtr(\_\_uuidof(ManagedClass));

 //rmgLibraryPtr rmgPtr;

 string inputData = "";

 bool toEnd = false;

 int row;

 int col;

 string temp;

 long \*result = new long;

 long item;

 //BSTR direct;

 //Start the servers

 rmgPtr->StartConnection();

 //rmgPtr->MoveDown();

 while(toEnd != true)

 {

 cout << "Please enter a command or quit to end" << endl;

 cin >> inputData;

 if(inputData == "quit")

 {

 toEnd = true;

 }

 else if(inputData == "up")

 {

 rmgPtr->MoveUp();

 }

 else if(inputData == "down")

 {

 rmgPtr->MoveDown();

 }

 else if(inputData == "left")

 {

 rmgPtr->MoveLeft();

 }

 else if(inputData == "right")

 {

 rmgPtr->MoveRight();

 }

 else if(inputData == "movLoc")

 {

 cout << "Please enter the row";

 cin >> row;

 cout << "Please enter the column";

 cin >> col;

 rmgPtr->MoveSetLocation(row,col);

 }

 else if(inputData == "checkWall")

 {

 cout << "Please enter a direction (0 = up, 1 = down, 2 = left, and 3 = right";

 cin >> item;

 if(item == 0)

 {

 rmgPtr->CheckWallUp(result);

 }

 else if(item == 1)

 {

 rmgPtr->CheckWallDown(result);

 }

 else if(item == 2)

 {

 rmgPtr->CheckWallLeft(result);

 }

 else if(item == 3)

 {

 rmgPtr->CheckWallRight(result);

 }

 else

 {

 cout << "Nothing Sent Please enter one of the correct values." << endl;

 }

 //rmgPtr->CheckWall(item,result);

 /\*if(\*result == 0)

 {

 cout << "WALL" << endl;

 }

 else if(\*result == 1)

 {

 cout << "PATH" << endl;

 }\*/

 cout << \*result;

 }

 else if(inputData == "curRow")

 {

 rmgPtr->GetCurRow(result);

 cout << \*result;

 }

 else if(inputData == "curCol")

 {

 rmgPtr->GetCurCol(result);

 cout << \*result;

 }

 else if(inputData == "endCheck")

 {

 rmgPtr->CheckEnd(result);

 cout << \*result;

 }

 //rmgPtr->ReadMessage();

 /\*switch(inputData)

 {

 case "quit":

 case "Quit":

 toEnd = true;

 break;

 case "up":

 rmgPtr->MoveUp();

 break;

 case "down":

 rmgPtr->MoveDown();

 break;

 case "left":

 rmgPtr->MoveLeft();

 case "right":

 rmgPtr->MoveRight();

 case "movLoc":

 cout << "Please enter the row";

 cin >> row;

 cout << "Please enter the column";

 cin >> col;

 rmgPtr->MoveSetLocation(row,col);

 break;

 case "wallCheck":

 cout << "Please enter a direction";

 cin >> temp;

 rmgPtr->CheckWall(temp);

 result = rmgPtr->ReadMessage();

 }\*/

 }

 rmgPtr->EndConnections();

 /\*TestConnectorPtr t1(\_\_uuidof(ManagedClass));

 t1->StartServer();\*/

 return 0;

}

**sManagedDLL:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

//Tutorial from http://support.microsoft.com/en-us/kb/828736

//NOTE: THE KEY FILE CAN BE CREATED FROM THE PROPERTIES OF THE MAIN PROJECT!!!!!

namespace sManagedDLL

{

 //interface declaration

 //Needed in order to declare the class

 public interface ICalculator

 {

 int Add(int Number1, int Number2);

 };

 //Interface Implementation

 public class ManagedClass:ICalculator

 {

 public int Add(int Number1, int Number2)

 {

 return Number1 + Number2;

 }

 }

}

**CPP Client:**

// CPPClient.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

// Import the type library.

#import "..\sManagedDLL\bin\Release\sManagedDLL.tlb" raw\_interfaces\_only

using namespace sManagedDLL;

int \_tmain(int argc, \_TCHAR\* argv[])

{

 // Initialize COM.

 HRESULT hr = CoInitialize(NULL);

 // Create the interface pointer.

 ICalculatorPtr pICalc(\_\_uuidof(ManagedClass));

 long lResult = 0;

 // Call the Add method.

 pICalc->Add(5, 10, &lResult);

 wprintf(L"The result is %d ", lResult);

 // Uninitialize COM.

 CoUninitialize();

 return 0;

}

**Connection Test:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.IO.Pipes;

using System.Threading;

//Tutorial from www.dijksterhuis.org/using-named-pipes-in-c-windows/

namespace ConnectionTest

{

 public interface TestConnector

 {

 void StartServer();

 void StartClient();

 //void TestSend();

 };

 public class ManagedClass:TestConnector

 {

 public void StartServer()

 {

 using (System.IO.Pipes.NamedPipeServerStream pipeStream = new NamedPipeServerStream("TestPipe"))

 {

 Console.WriteLine("[Server] Pipe Created {0}", pipeStream.GetHashCode());

 Console.WriteLine("Waiting for client connection");

 //wait for connection using a loop

 pipeStream.WaitForConnection();

 //while (pipeStream.IsConnected == false)

 //{

 // //busy loop

 //}

 //if here then connected

 Console.WriteLine("[Server] Pipe Connection Established");

 using (StreamReader sr = new StreamReader(pipeStream))

 {

 string temp;

 //read line from pipe and print with current time

 while ((temp = sr.ReadLine()) != null)

 {

 Console.WriteLine("{0}: {1}", DateTime.Now, temp);

 }

 }

 pipeStream.Close();

 }

 }//End of StartServer

 public void StartClient()

 {

 //Start client only after server has created the pipe

 // ManualResetEvent SyncClientServer = (ManualResetEvent)obj;

 using (NamedPipeClientStream pipeStream = new NamedPipeClientStream("TestPipe"))

 {

 //connect to server

 pipeStream.Connect();

 Console.WriteLine("[Client] Pipe Connection Established");

 using (StreamWriter sw = new StreamWriter(pipeStream))

 {

 sw.AutoFlush = true;

 string temp;

 Console.WriteLine("Please type a message and press [enter], or type quit to exit the program.");

 while ((temp = Console.ReadLine()) != null)

 {

 if (temp == "quit")

 {

 break;

 }

 sw.WriteLine(temp);

 }

 }

 pipeStream.Close();

 }

 }//End of StartClient

 }

}

**TESTDLLCONNECTION:**

// TESTDLLCONNECTION.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

//import the library

#import "..\ConnectionTest\bin\Release\ConnectionTest.tlb" raw\_interfaces\_only

using namespace ConnectionTest;

int \_tmain(int argc, \_TCHAR\* argv[])

{

 //Start the Server Connection

 HRESULT hr = CoInitialize(NULL);

 TestConnectorPtr t1(\_\_uuidof(ManagedClass));

 return 0;

}

**Journal/Blog**

5/12/2015

It is now the final day to work on the project seeing as all material will need to be submitted today. The

past few days I have worked on updating the website some and worked on creating a user manual for the program.

It has been a good year and I hope this information can be helpfull in some way to future computer science

students.

5/9/2015

Classes are now officaly over leaving only finals. Now that the presentations and defenses are over the only

thing left for this project is to submit all of the materials used, code created, and create a user manual for

the software.

4/11/2015

Since my last update I have added some more of the main features to the maze generator. These features

include the ability to move through the maze and to log each move in a list box, which will be used

to help track the step by step movement through the maze. The movement is covered by a series of different

functions each for a different direction as well as one function that will allow placement to a specific

point in the maze. While I do still have a few things to add to the generator such as the help screen and

the creation and saving of a maze I have started working on a way to connect the maze generator to a c++

program in order to allow the user to create the recursive functions that will traverse the maze. For this

I have looked into creating a dll which will hold the functions that the c++ program will use in order to

traverse the maze. I will also use this dll in order to set up the link between the two programs.

3/8/2015

As we start moving into the second half of the semester we are about to enter the project walkthroughs

in which we will be showing what we have of our projects to the rest of the class. While not quite a

presentation this will help prepare us for the presentations at near the end of the semester as well

as a way to obtain some feedback and advice about what we have done with the project so far. This

past week I have worked on finishing the loading and displaying of the mazes. While I have not

implemented the movement through the maze it should not take very long to do.

3/1/2015

This week I met with Dr. Pankratz twice to go over what I had for my project and to go over some

ideas about how to move forward with it. This weekend I have started creating the Maze generator

interface and have started programming the generator to open files for completed mazes. This is

being done before the maze creation mechanism is added so that the file input and output system

is more complete and so that I can have a way to save and load the completed mazes. Currently

the load system is still being made and should be completed by Tuesday March 3, 2015 at the latest.

2/22/2015$

This past week I did a bit more research on api and have started thinking of the interface that I

will have for the maze generator. I have also updated parts of the website so that

2/16/2015$

For this past week I have mostly focused on researching API and how to link a c# program to c++ in

order to transfer data about the maze from differenet programs. I have also created a timeline

in which I hope to follow in order to complete the project by the end of the semester. This will

be uploaded onto the site soon as will several other updates.$

2/5/2015$

The computer science majors of the class of 2015 have now received there projects and have started

working on the research that is involved in creating their systems. This first entry in the

progress blog is to let you know some general information about the capstone project and what I have done this first week on my own project.

For this the Computer Science Capstone Project each student is assigned a project idea that they

must then use to create a functional system from by the end of four months.

The description of the project that was given can be found by clicking the Project Info

link at the top of the page for those that want to see the exact description that was given

for this specific project. This description gives a sentance or two about what the project

is about as well as about five things that should be included in the final system.

As well as an idea on what might be good to include. For my project I am to create a

way to visually show recursion to the user through the use of a maze. With the program being able to

connect to recursive programs that have been created in various languages.

For this first week I have mostly thought of different ways that the program can go, such as using C# to create the visual maze. So far little research has been done for the project. I have also started the website that will show my progress in the project which will be updated as the project continues.

**Resume**

**Philosophy**

For most of my life I did not have a computer that I could use regularly. Rather once computers

began to be used more than books I would always go the library to use computers. Due to this

I never thought to work with them later in life and actually planned to go into the field of

veterinary medicine for most of it. This changed during the second semester of high school

when I took an electronics class where one of the assignments in class invloved programming

a small robot. I thoughouly enjoyed this and found myself taking classes in college for

computer science rather than biology based ones.

Though my years at college I have learned not only one language other than english but several.

Of these only Japanese is a normal language which is spoken by humans leaving the rest as

languages that are involved in programming. I have found that while these languages might

seem strange to most people they can be extremly useful to those that know them and are

needed to make the various software that poeple use on computers and other electronic devices.

From what I have seen computer science and programming are tools which allow people to create

devices which can aid many people throughout their lives. To me these tools allow us to create

various programs but also as a way to better the future.

**Career Plans**

 As of May 11, 2015 my career plans are a little vauge. Eventually I hope to be able to live and work in Japan, however I do not have any real method of getting there. For now I seek to find a job with which I can use to pay off my school debts and pay any living expenses that I need covered.

**Sources**

 For this project I used a variety of sources. The most basic of which was the information gained through classes in the last four years. Other than that I gained some insight on the design aspects of the project by meeting with Dr. Pankratz a few times and from Dr. Mcvey during the project walkthroughs. Otherwise I used the internet to research API and various function syntax. For these I used a lot of msdn for the basic c# information and stack overflow for some information on how to implement certain aspects. I also reasearched a lot about pipes and how to connect two programs. Below is a list of the sites that I used in the research as well as what I used for general information.

"C# DllImport and Dllexport: DLL Interop." C# DllImport and Dllexport: DLL Interop. Web. 12 May 2015. <http://www.dotnetperls.com/dllimport>.

"Fernando Zamora." : Simple Example of Using Pipes with C#. Web. 12 May 2015. <http://fernandozamorajimenez.blogspot.com/2011/10/simple-example-of-using-pipes-with-c.html>.

"How to Add Dll in C# Project." - Stack Overflow. Web. 12 May 2015. <http://stackoverflow.com/questions/7685718/how-to-add-dll-in-c-sharp-project>.

"How to Call a Managed DLL from Native Visual C Code in Visual Studio.NET or in Visual Studio 2005." How to Call a Managed DLL from Native Visual C Code in Visual Studio.NET or in Visual Studio 2005. Web. 12 May 2015. <http://support.microsoft.com/en-us/kb/828736>.

"How to Create a DLL Library in C and Then Use It with C#." - CodeProject. Web. 12 May 2015. <http://www.codeproject.com/Articles/9826/How-to-create-a-DLL-library-in-C-and-then-use-it-w>.

"How To: Create and Use C# DLLs (C# Programming Guide)." How To: Create and Use C# DLLs (C# Programming Guide). Web. 12 May 2015. <https://msdn.microsoft.com/en-us/library/3707x96z(v=vs.90).aspx>.

"How To: Use Named Pipes to Communicate Between Processes over a Network." How To: Use Named Pipes to Communicate Between Processes over a Network. Web. 12 May 2015. <https://msdn.microsoft.com/en-us/library/vstudio/bb546085(v=vs.100).aspx>.

"Inter-Process Communication between C# and C Using Named Pipes." - CodeProject. Web. 12 May 2015. <http://www.codeproject.com/Tips/420582/Inter-Process-Communication-between-Csharp-and-Cpl>.

"Inter-Process Communication in .NET Using Named Pipes, Part 1."Inter-Process Communication in .NET Using Named Pipes, Part 1. Web. 12 May 2015. <http://www.codeguru.com/csharp/csharp/cs\_misc/sampleprograms/article.php/c7259/InterProcess-Communication-in-NET-Using-Named-Pipes-Part-1.htm>.

"Inter-Process Communication in .NET Using Named Pipes, Part 1." - CodeProject. Web. 12 May 2015. <http://www.codeproject.com/Articles/7176/Inter-Process-Communication-in-NET-Using-Named-Pip>.\

"Interprocess Communication Using Named Pipes in C#." Tech.pro. Web. 12 May 2015. <http://tech.pro/tutorial/633/interprocess-communication-using-named-pipes-in-csharp>.

"Introduction to Win32 Named Pipes (C )." Avid Insight. Web. 12 May 2015. <http://avid-insight.co.uk/2012/03/introduction-to-win32-named-pipes-cpp/>.

"Stack Overflow." Stack Overflow. Web. 12 May 2015. <http://stackoverflow.com/>.

"Using Named Pipes in C# / Windows - 1." Martijns C Programming Blog RSS. Web. 12 May 2015. <http://www.dijksterhuis.org/using-named-pipes-in-c-windows/>.

"Using Named Pipes to Connect a GUI to a Console App in Windows." Dr. Dobb's. Web. 12 May 2015. <http://www.drdobbs.com/windows/using-named-pipes-to-connect-a-gui-to-a/231903148>.

"Writing a Simple Named Pipes Server in C#." Writing a Simple Named Pipes Server in C#. 3 Oct. 2012. Web. 12 May 2015. <http://www.dzone.com/articles/writing-simple-named-pipes>.