AQA qualification training

A-level Computer Science

Focus on Non Exam Assessment

BOOKLET 5

Published date: Summer 2016 version 1.0

MazeSolving Project

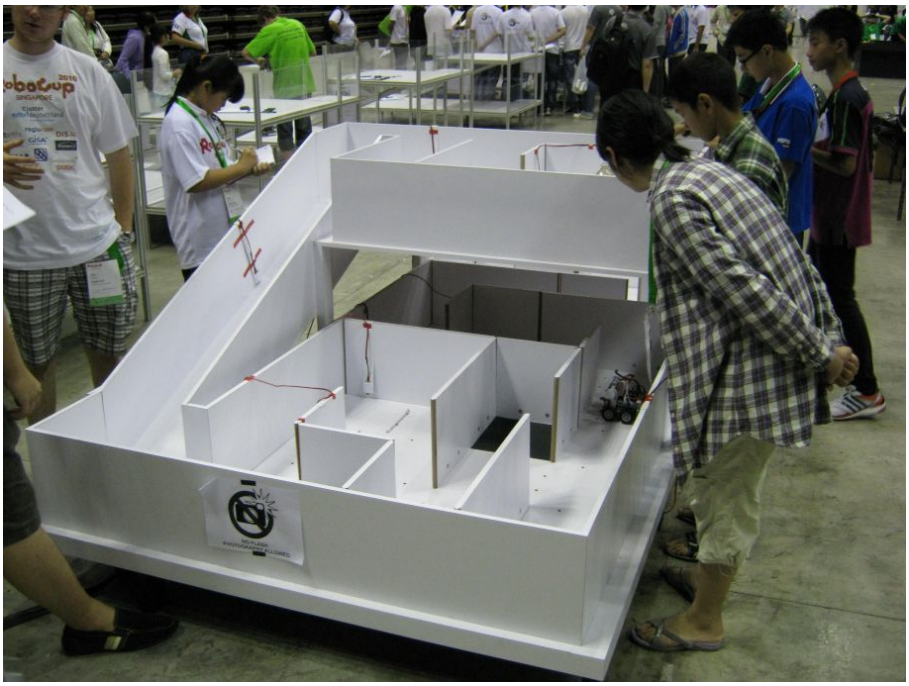
AQA Exemplar

Contents

Analysis

Problem Background

A school robotics club has now been running for a number of years and one of the most interesting challenges for any student at the club is the RoboCup Junior (www.robocupjunior.org/) competition. Although it has many parts including soccer, dance and rescue, the club has a particular interest in the rescue competition. The rescue competition is split into two different categories, rescue A and rescue B, both challenging events. In rescue A the student's robot must follow a line, avoiding obstacles and moving a can at the end. Rescue B is even more difficult with robots finding their own way around a maze, discovering thermal sources, avoiding objects, avoiding black squares, travelling up a ramp and stopping in the same position as the robot started in.



Rescue A arena

The current rules for rescue B are included in Appendix A, with a summary below.

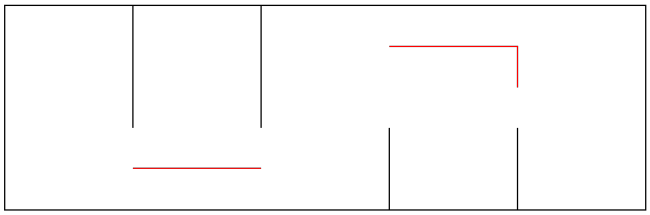
Robocup Junior Rescue B Rules Summary

In this event, a robot must traverse a maze of tiles 30cm by 30cm. The maze is random and has an upstairs and downstairs. The aim of the robot is to travel around the maze, detecting as many thermal sources as possible and then returning to its starting position. Since the robot can only be in the maze for 8 minutes, it is important to find the most efficient way to cover all of the cells. In doing this it guarantees that every cell and so every wall is checked for thermal sources, as so every source is found. The final score is based on the number of sources found with a bonus for returning to the starting position.

Students attempting to enter rescue B have found that one of the most significant challenges faced is the problem of finding an algorithm to guarantee that their robot travels past every wall at least once, while completing the maze in the least time possible and stopping in the same place as the robot started. To make this harder still, the maze always features 'floating' walls, walls not accessible by just following one wall. Following the left hand wall would get the robot back to the start but would not access every wall. The thermal sources can be located on any wall, on either side. Points are awarded based only on the number of thermal sources detected, with a multiplier of this value if you get back to the start. As a result of this it is important that the robot must access every cell in the maze, while still finishing within time constraints. The robot only has a set period of time to complete the maze.

Floating Walls

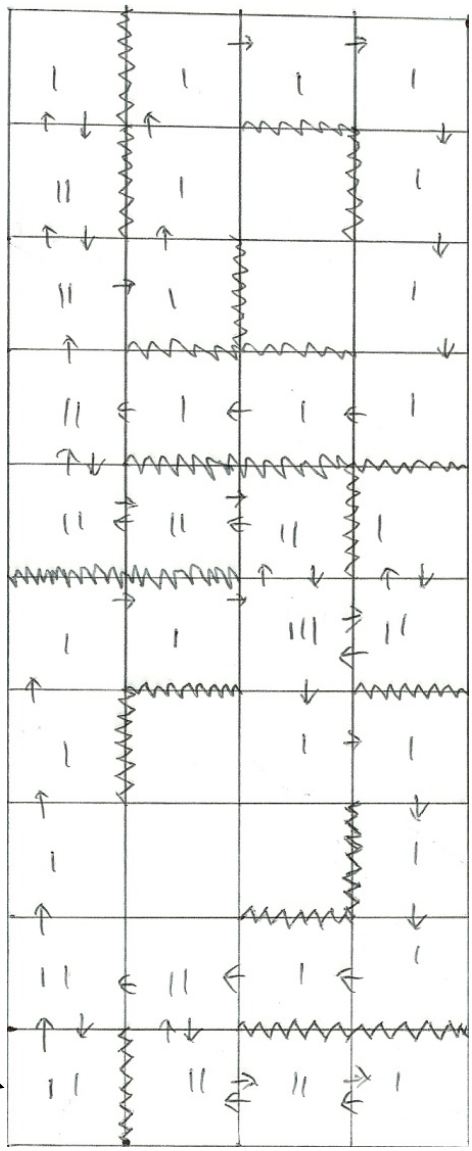
A floating wall is a wall that cannot be accessed by following a single wall all the way round. These walls can be joined together, but the set of walls is not linked to the perimeter of the maze.



Floating walls are highlighted in red.

Currently to find the most efficient algorithms these students express their ideas through pen and paper, counting the individual number of moves that would be taken so as to give a measure of efficiency.

The path is represented by arrows, but it is also unclear



The number of dashes represents the number of times visited

The start is here, but it is unclear

This is clearly not the most efficient method of testing algorithms and often the results of this testing are inconclusive due to the limited number of mazes tested and the errors that occur in trying to draw the path on a hand drawn maze while counting the number of moves. This method could be improved by printing off mazes, but this is still inefficient with mazes still being designed from scratch and paths still being drawn by hand.

A computerised system that could generate random valid mazes and then test a robot navigating this maze with various algorithms would be very beneficial to the students and the clubs. This is the problem that I will be attempting to solve.

<u>Possible Solutions</u>

1. Manual system:
    i. Drawing the mazes by hand: This solution is time consuming and can lead to information being lost easily, as well as producing a lot of errors.

ii. Using excel to create the maze design: This is quicker but still requires the user to design mazes themselves.

2. Using a system to generate valid mazes that users can explore virtually. This solution allows very quick generation of mazes, storing of previous solutions, and an indication of efficiency can be calculated. This solution would greatly speed up the process of testing algorithms:
    i. A web based design could be used, this would allow the system to be accessed anywhere and information stored would be located on the web server. A web based solution would also run on far more devices, it would not be necessary to compile a number of different files for different devices, as a non-web based solution would require. I would use HTML to produce this solution, as it would allow access to the <canvas> tag, JavaScript could then be used to generate the maze.
    ii. A non-web based design would not require memory space on a server, and files could be stored in a memory of a user's computer. This would also enable users to explore mazes when they are offline. User would be in charge of managing these files, and would need to be made aware of this.

I feel that the most appropriate solution is solution 2i, since it would make the system more accessible to end users. In addition to this, it would also allow access to features offered by HTML5, most importantly the canvas tag and the functions that are supplied to draw lines onto this canvas.

Identification of end-users

This system would be used by the students at the club to help identify the most efficient algorithms, and if very successful could also be used by students from other schools also enter the competition. NJ, who assists in running robotics club for students, and will be a key end user using this project to assist those pupils comparing algorithms.

The system will also be used by the computing teachers as it will provide a method of projecting maze solving algorithms onto the screen and also discuss complexity.

Acceptable Limitations

The mazes generated will be based around the physical structure of the current Rescue B maze installed at A School (this came from the 2013 World Finals). Only the bottom section of the maze is going to be modelled, and the layout of this section is a 4x10 array of squares. These limitations have been discussed with the end users, and were accepted and agreed to.

Objectives

The first objective of this system would be to produce a random valid maze. To ensure that the mazes produced are valid, all the points in the maze need to be accessible by the virtual robot. This would mean that no areas of the maze would be wasted, e.g. walls not stopping access to one half of the maze.

1. The maze would also need to be generated with at least one floating wall to comply with the rules of the competition.

2. The second objective would be to allow users to test either own algorithms by manually exploring one of the random mazes. They will be able to move their robot around the maze on the screen, imputing each move and watching it move from cell to cell.
3. The number of moves being counted as it moves through the maze will be counted to give an idea at the end of how efficient the algorithm was, comparing it to the optimal solution which would need to be found.
4. The next objective is to allow, using an inbuilt algorithm, the user would be able to see the efficiency of these algorithms to compare to their own. A number of algorithms could be simulated in this system (for example – random walk, …). The system would show the robot moving around the maze on its own(with delays between moves), giving reasons for each move to the user (e.g. North and south are open, just came from north, going south).
5. A method of storing mazes and the current best number of moves they can be solved in and previous paths taken.
6. A method of allowing the user to input their own maze design, i.e. clicking where they would like their walls.

End user feedback

Having looked at the objective of the Project, It is clear to me that the limitations will not alter the overall functionality of the project. I believe this will still enable us to use the system to test Mazes and maze solving algorithms.

Through research and development he will enable us to test algorithms from within his Software, so that we will be able to use the Algorithms more effectively within our Robotics Projects.

Interview 1

Jonathan, a computing student and member of the schools rescue B team, is interviewed:

Do you feel that a computerized system would be beneficial to determining the efficiency of algorithms and would you find this tool useful?
Yes, this would take a lot less time than if we were to use excel or a piece of paper. A feature that I would include is a count to find out how many moves a robot has done, as we spent a lot of time last year arguing about which algorithm was the best.

What other features would your find useful?
I would like some way to track progress around the maze, with cells highlighted depending on how many times they have been visited.

Would you like information to be saved, such as previous mazes?
Yes, some sort of graph or chart to compare different mazes or runs, again showing how many moves taken, maybe taking turns into account by having arbitrary 'score' rather than a number of moves.

Interview 2
Carl, another student in the team, is also interviewed:

To what extent do you think a computerized system would help you design a maze solving algorithm?
For me, a computerised system would mean that theories can be tested infinitely and rapidly. The fact that it will have easy-to-use visual representation, as well as not having the hassle of needing to be drawn out each time, would make it a much easier to further develop an algorithm.

What features would you find most useful?
I think perhaps the option of manually being able to take control of the robot within the virtual maze. However, there could be some benefit in having an automatic version, e.g. in travelling the shortest distance from point X to point Y. The option of having a randomly generated maze will help not only in shortening the time needed to test an algorithm and the time taken to draw out possibilities, but also the fact that it will mean our algorithms can be tested multiple times in a multitude of different scenarios.

Would the ability to use the system online and from any location benefit you, more than an offline system that would not require an internet connection?
I'd prefer something that didn't require the internet; any downloadable application to a variety of different platforms would be beneficial, though for me, I'd say especially a downloadable PC application as that would mean users could use the keyboard, have the easier-on-the-eye GUI and all bits of information can be made visually clear, e.g. in stats monitors on the sides or a neat gyro display.

How many mazes would you need to save over the course of one year?
Ideally, it should only be used in development stages. I can only see further usage of algorithm-aiding software being of benefit if you could simulate different robot aspects, such as wheel power, robot size and sensors, etc.

Would you want the option to turn floating wall on and off, or would you rather they were permanently on?
Definitely on and off. It would make it very easy to determine how well different algorithms stack against each other.

How many inbuilt algorithms would you like to be able to test?
Ideally as many as possible, though there are countless possibilities out there and knowing you, you will never be able to find more than a few others than you already know about. However, I'd definitely suggest keeping the option of manual movement through the maze.

Interview conclusions

The interviews have highlighted additional features that could also be included in the project, such as highlighting cells in different colours based on the number of times that they have been entered. Another interesting idea was that of giving solutions a "score", giving points to turns and moves forward, not just a point for a movement in any direction of one unit of distance. It would also be beneficial to turn floating walls on and off, as highlighted in the second interview. I disagree that the best solution for this project is non-web based, although preferable to the user. I disagree because if I was to produce an offline solution, I would be limited by time constraints to only one platform, and

this would limit the use to the end user. However, I do agree that the best possible solution would be offline and available in many platforms, perhaps even available as a Smartphone app.

Observations

As part of my analysis, I decided to visit the team one Friday after school, when the algorithm was being worked on. It seemed to be that the majority of the comparison of algorithms was done by explaining how the algorithm worked and then testing it on the whiteboard. In order to test a solution, a maze was carefully drawn onto the board and then a number of solutions were drawn on top of each other. It seemed to be that the generation of the mazes seemed to be the task that took most of the time. I observed that if they could directly test, skipping this step, a huge amount of time could be saved. This time could then be spent developing the robot or on further algorithm comparison. The team seemed to share this view, and suggested that storing mazes would be very helpful, as only one or two mazes could be drawn on the board at once. The other issue came in trying to count the number of moves, and keep track of the path taken by each algorithm. They appeared to have issue with the maze on the whiteboard getting very messy, and since most solutions take over 50 moves, there was often a counting error.

Data Dictionary

The end user will see the maze currently on the screen, with a number generated to represent this maze and the number of solutions currently stored for it. In addition, the path currently taken would be also represented as a number, once complete this number will be moved to become part of the number representing the maze. This minimal amount of data is another reason that a web-based solution is appropriate, as bandwidth in a web-based solution will never be an issue. The maze could be initially stored in cookies, but at a later stage this information could be stored in a server to allow user information to be permanently stored.

Data storage

Each maze could be stored as a string with a '1' or '0' to indicate if a wall is turned on or off. Since there are 66 walls in each 4 by 10 maze, ignoring any walls that would overlap with the outer wall of the maze. Since each wall takes only 1 bit of information, a maze could be stored in 9 bytes. However, other information must also be stored, such as the user path. If a path is a string of coordinates, with each coordinate consisting of 2 numbers so 2 bytes, and a average path being anything up to 100 steps long, we can say this information would take up to 200 bytes of information. Although this is only in the case of the most efficient storage, we know that a cookie can take anything up to 4096 bytes of information[1]. This tells us that it is very unlikely that any information that needs to be stored would be too large to store in a cookie. Since this is a very small amount of data, we can conclude that a cookie would be an appropriate storage method for the data produced in this project.

[1] How Big Can a Web Cookie Be? : http://webdesign.about.com/od/cookies/f/web-cookies-size-limit.htm

Documented Design

Overview

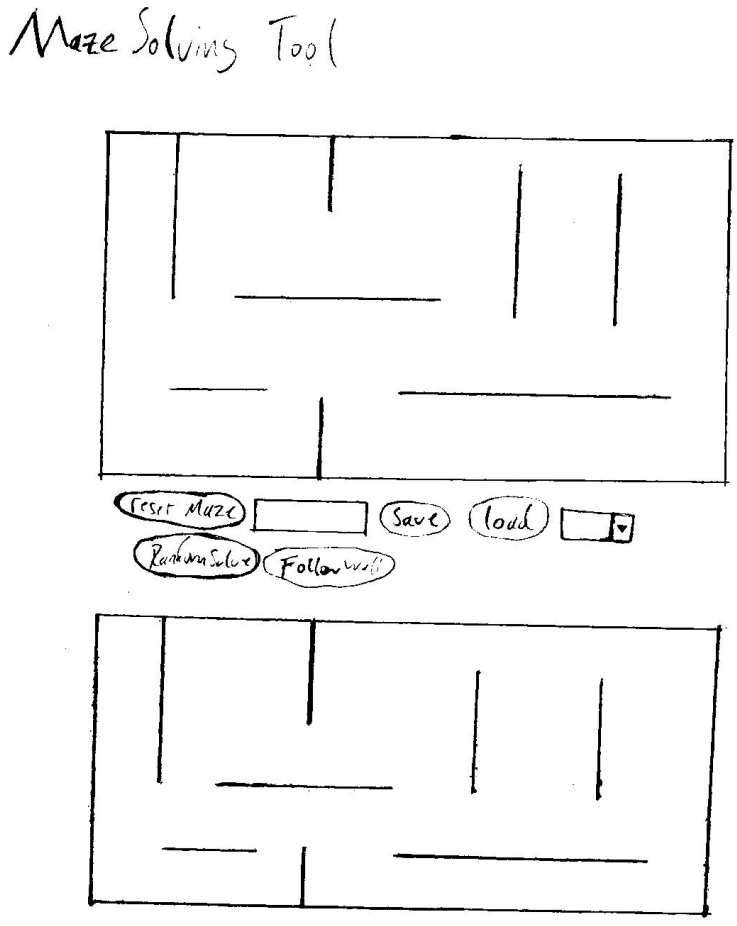
The aim of this project is to create a tool to allow the user to generate a random valid maze and then explore the maze using a virtual counter that can be moved around the maze. Validation would be required to only allow valid moves to be made, these are moves that do not allow the counter to pass through walls. Having done this, the next stage is to allow mazes to be saved and loaded, and at least one inbuilt algorithm will be available for the user to test, to compare to their own solution.

Description of modular structure of system

This project will require the following structures:

1. Maze Generation
2. Counter to move around the maze
3. Saving and loading features
4. Algorithm to reset maze
5. Inbuilt algorithms

User Interface



Individual cell layout and information

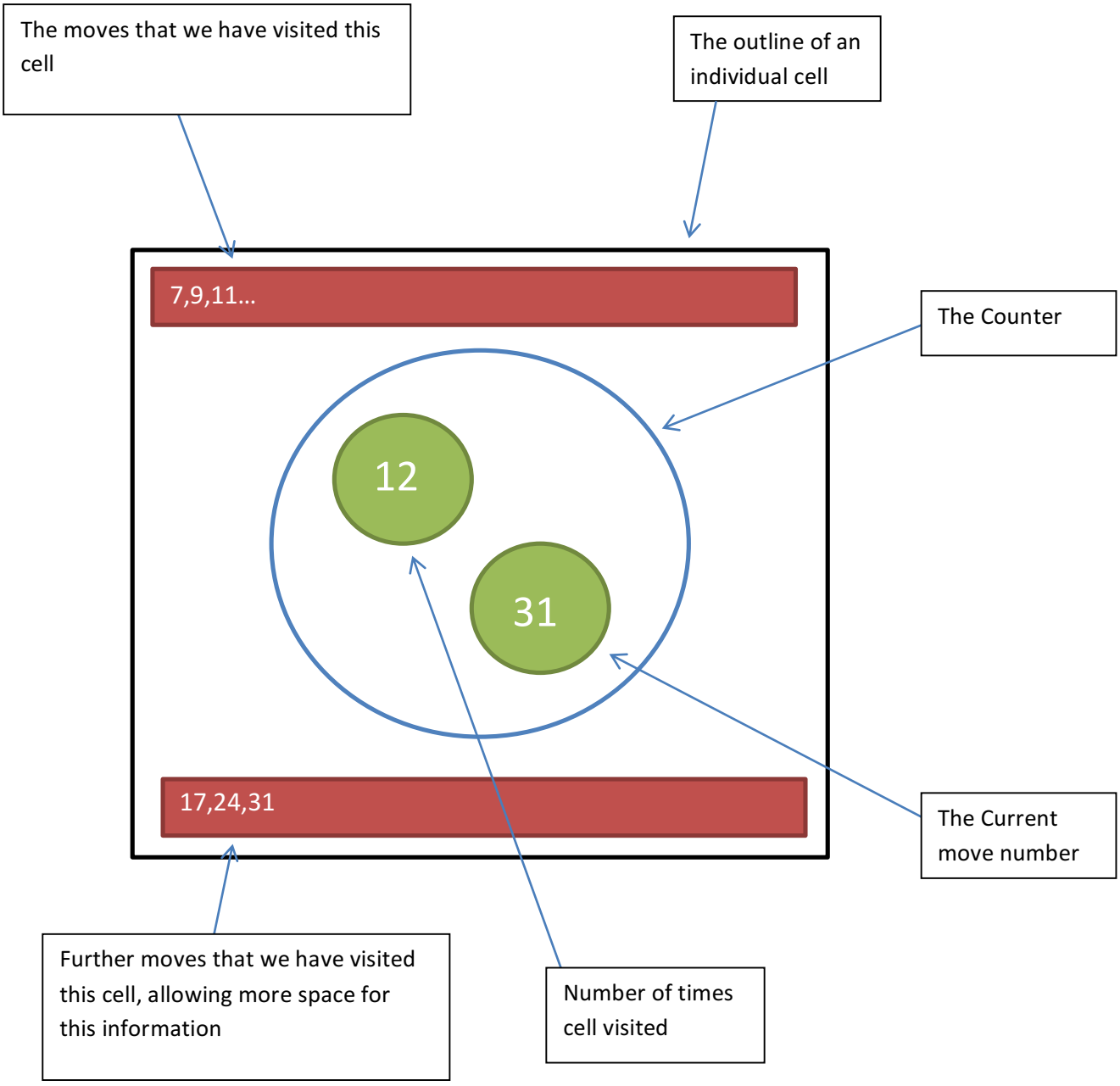


Table of Main Steps

Stage Number	Description
1	First prototype, fixed maze generated
2	Maze created by randomly turning walls on and off
3	First attempt to generate a valid random maze, bug present at this stage preventing generation.
4	The bug is fixed and a valid random maze is drawn.
5	First attempt to turn floating walls on and off, only able to reduce the number of floating walls.
6	Users are now able to explore the maze, as a counter is added for users to move round the screen, information such as path taken and move numbers are shown.
7	The path taken by the user is now shown on a second identical maze below the first, making the path taken more easily followed. A button to allow the maze to be reset is also added at this stage.
8	The first attempt at getting the maze so save and load is made, although unsuccessful at this point.
9	The saving feature is not yet completed, an attempt has been made to fix it.
10	At this stage saving and loading feature has been completed.
11	Saving and loading has been improved, more than one maze can now be saved, with users naming their mazes and being able to select their previously saved solutions from a dropdown list. This information is stored in a cookie.
12	The first attempt is made to add a random solve feature, randomly solving the maze using a pre-set algorithm.
13	Random solve now works and there is now a button to turn floating walls on and off.
14	A first attempt is made to add a follow wall procedure, however this is unsuccessful at this stage.
15	The page is given a new layout, to look more like the user interface first designed.
16	In this final stage, the follow wall procedure is successfully added.

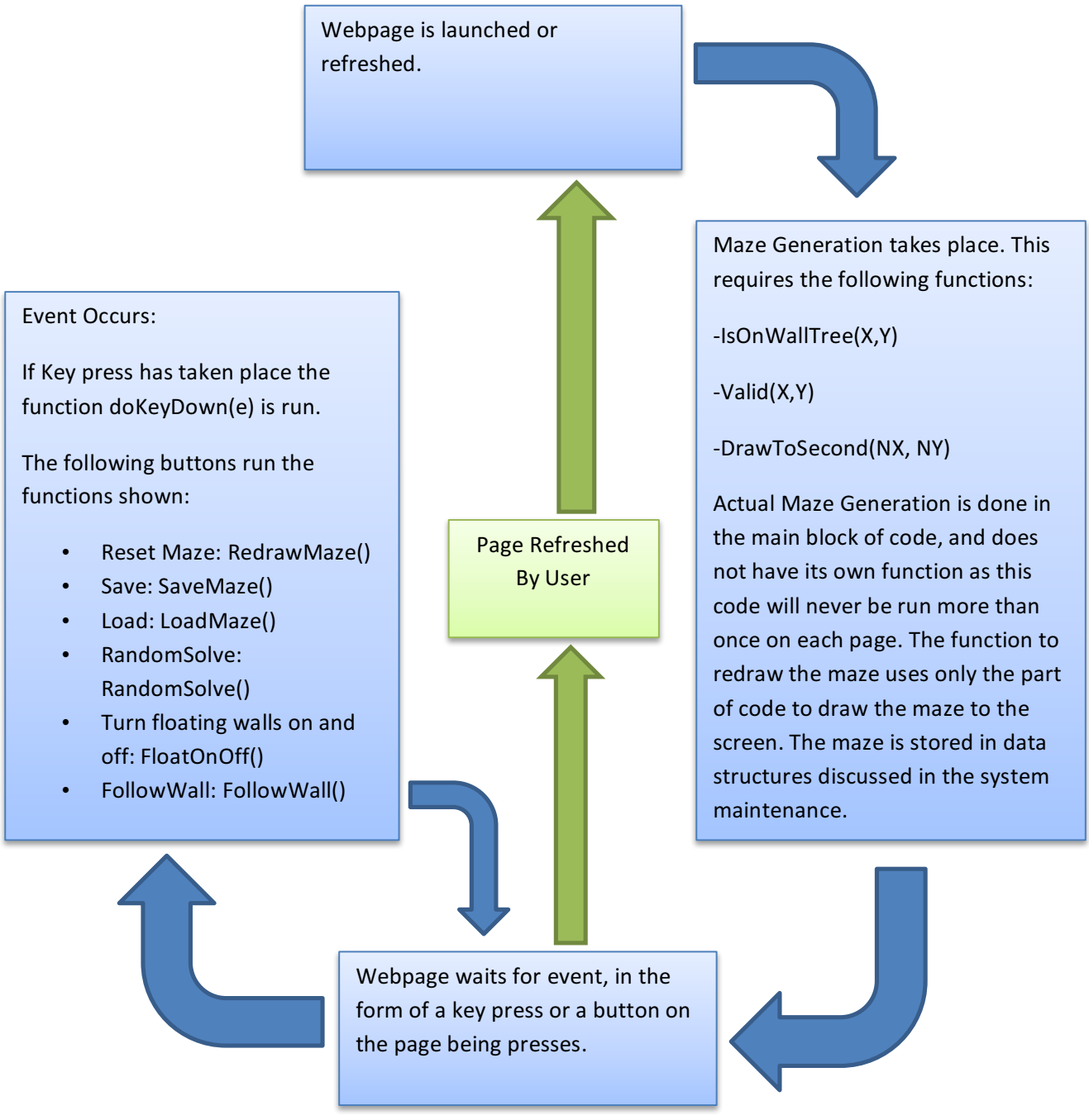
Table of Functions

A full function list and descriptions can be found in the system maintenance section, this table shows which functions are added at each stage.

Stage	Function(s) added
4	• IsOnWallTree(X,Y), Returns true if the inputted point is part of a tree that is connected to the wall. • Valid(X,Y), This function will check a wide range of criteria to decide whether or not the point about to be added to the current tree is allowed to be added to the current tree. This includes things like whether it is inside the canvas, whether it crosses another tree attached to the wall when it itself is part of a tree attached to the wall, etc.
6	• AddNextMove(),Adds the current move to the top or bottom of the cell the counter is currently in, adding to the list that is already there or starting one if there is not.

	• doKeyDown(e), This function runs when the user enters a key press. • FloatOnOff(), switches the FLOATING variable to false if it is true, and true if it is false.
7	• RedrawMaze(),Resets the current maze. • DrawToSecond(NX, NY), Draws a line to the inputted point from the last point inputted onto canvas2.
11	• ReadCookie(name), this function finds the cookie with the name inputted and returns its value. • setCookie(c_name, value, exdays), Sets a cookie with the name, value and duration inputted as parameters. • eraseCookie(name), This erases the cookie with the name inputted. • SaveMaze(), saves the maze under the name currently in the textbox. • LoadMaze(),loads the maze currently selected from the drop down box.
13	• RandomSolve(), runs the random solve algorithm.
16	• LeftOpen(dir)*, returns true if the wall in one position to the left in the direction entered, from the counter, is open, else returns false. • FrontOpen(dir)*, returns true if the wall in one position in the direction entered, from the counter, is open, else returns false. • MoveDir(dir)*, move the counter forward, in the direction given. • FollowWall(),runs the follow wall algorithm. *These functions are further discussed in the system maintenance section.

Webpage events Cycle



Maze Generation – introduction

The first step in creating the maze solving tool will be to generate a random valid maze to be solved by the user. My initial approach to this was to create a single fixed maze to show the format that the maze would be displayed in. The first step was to decide how the mazes would be generated, and with further research I decided that using the canvas tag that available in HTML5 I would be able to

use javascript to draw lines onto the maze to form walls. The functions provided to achieve this are .moveTo() and .lineTo(), used together a sketch can be drawn on the canvas. Once the maze is completed .stroke() is used to draw the lines in over the invisible sketch. The line drawing functions take coordinate inputs and the stroke function is needed to draw to screen.

.moveTo = Sets an initial coordinate for the line to be drawn from

.lineTo = Draws a line from the initial coordinate to the coordinate specified as entered into the function.

An example using these functions is shown below:

```
<html>
  <head>
    <title>Page title</title>
    <style type="text/css">
    canvas {border-style:solid;}
</style>
  </head>
  <body>
   <canvas  id="a" width="1000" height="400"></canvas>

   <script>
    canvas = document.getElementById("a");
    context = canvas.getContext('2d');

    context.moveTo(100, 200);
    context.lineTo(100, (100));

}
    context.strokeStyle = "#202020";
    context.stroke();
</script>
  </body>
</html>
```

This code shows a canvas being defined, and given an id, here the id is “a”. The canvas is given a solid border to act as outer walls. Within the script we then create two new variables. “canvas” is the variable we assign the canvas to, and “context” is the variable we assign the context of the variable to. The context assigns the number of dimensions that the canvas will have. As these variables are done using id it is possible to have more than one canvas and manipulate more than one canvas using javascript on one page. We then refer to the context variable when using the functions. This code draws a line from (100,200) to (100,100). These are pixel coordinates in the canvas. The line is stroked to draw it to the screen.

First Prototype – Stage 1

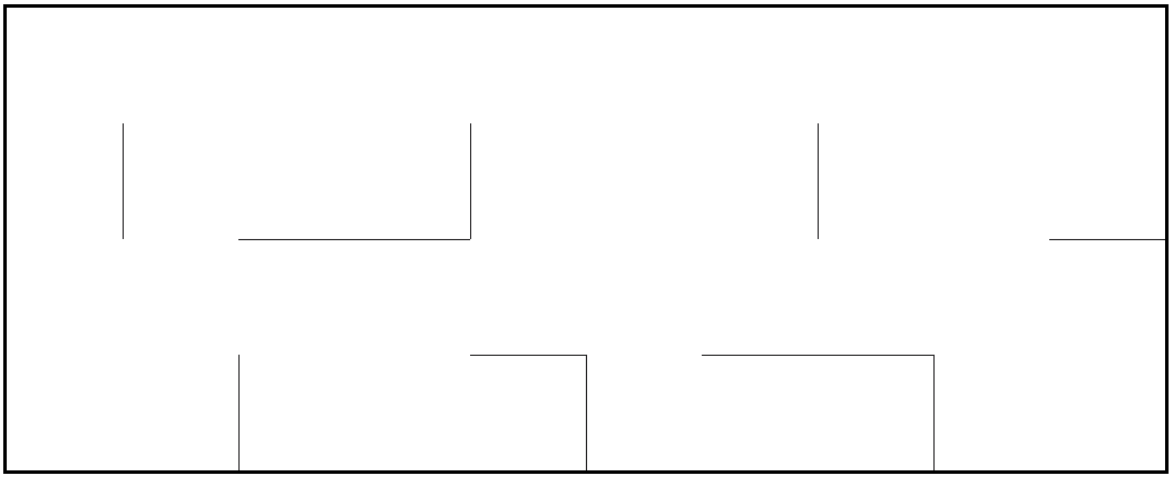
The aim of the first prototype is to generate a canvas onto which we draw a fixed maze. This is to confirm that the canvas tag could be used to draw a maze to the screen.

Pseudo code:

Generate a canvas tag with a border

Place some horizontal and vertical lines using .lineto

Stroke the canvas



```
<html>
  <head>
    <title>Page title</title>
    <style type="text/css">
    canvas {border-style:solid;}
</style>
  </head>
  <body>
   <canvas  id="a" width="1000" height="400"></canvas>

   <script>
    canvas = document.getElementById("a");
    context = canvas.getContext('2d');

    for (var x = 0.5; x < 1000; x += 100) {
    for (var n = 0; n < 400; n += 100){
    if(((n == 100) && ((x == 100.5) ||(x == 400.5) || (x == 700.5))) || ((n == 300) && ((x == 200.5) ||(x == 500.5) || (x == 800.5)))){
    context.moveTo(x, n);
    context.lineTo(x, (n+100));
    }
    }
}

    for (var y = 0.5; y < 400; y += 100) {
    for (var n = 0; n < 1000; n += 100){
    if(((y == 200.5) && ((n == 200) ||(n == 300) || (n == 900))) || ((y == 300.5) && ((n == 400) ||(n == 600) || (n == 700)))){
    context.moveTo(n, y);
    context.lineTo((n+100), y);
    }
    }



}
    context.strokeStyle = "#202020";
    context.stroke();
</script>
  </body>
</html>
```

Second Prototype – Stage 2

The second aim is to generate a random maze that is not necessarily valid. Other methods of storing the maze and generating it from storage are explored here. Using the function Math.random() to generate a random number from 0-1 then multiplication and Math.Floor I was able to generate random ones and zeros, each representing a different wall. A string of 80 digits is then generated, as show in the code below. This random string is then used to generate the maze.

```
var number = "";

for (n = 0; n < 80; n++){
number += (Math.floor(Math.random()*2)).toString();
}
```

The string is then translated into an array that can be used to generate the maze. The array structures are first generated, and then a mathematical function is used to find different digits in the string. These digits are then moved into the array structure to be easily accessed when generating the maze.

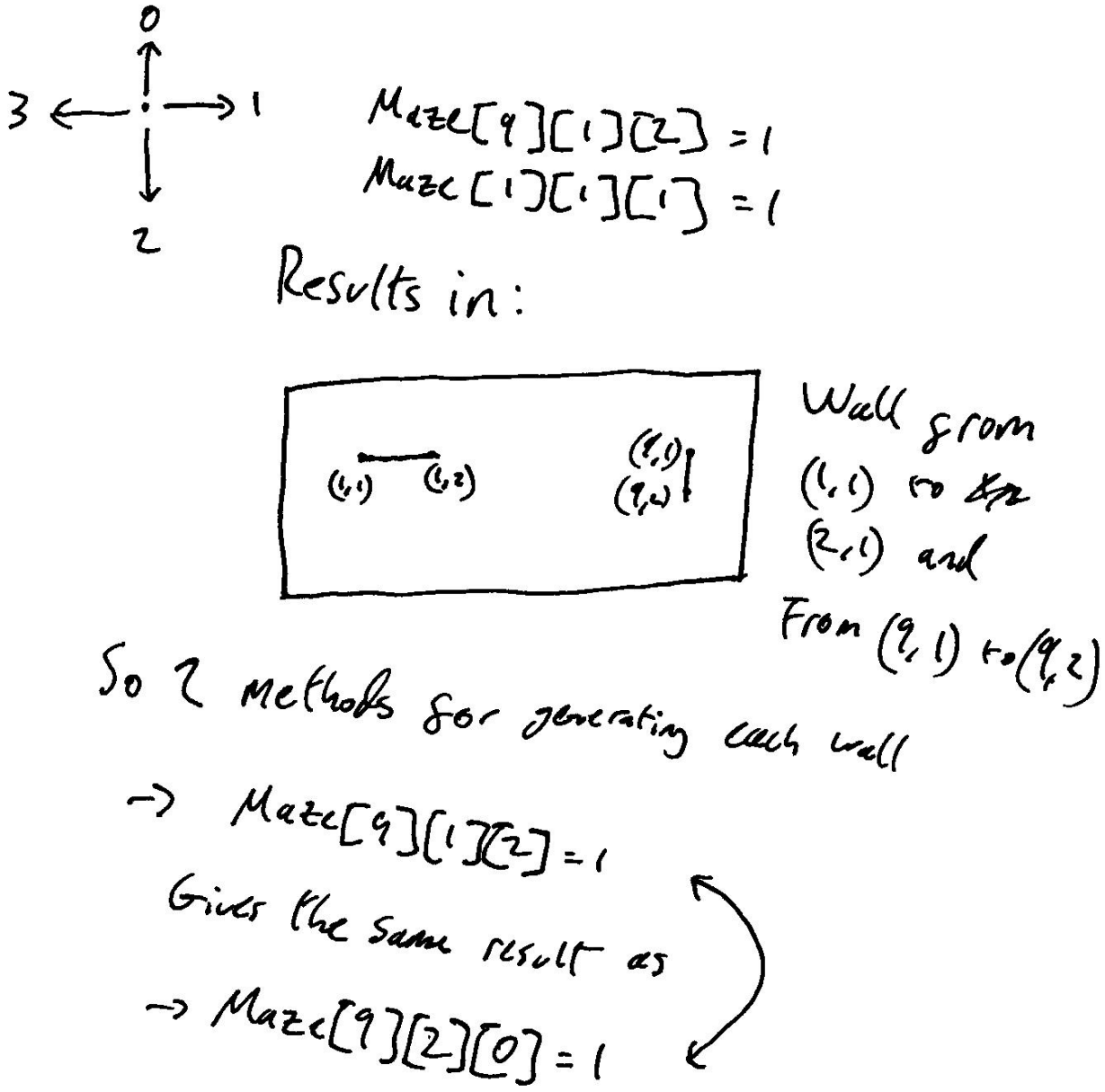
Creating Structures:

```
for (var i = 0; i < 10; i++) {
maze[i] = new Array();
}

for (var i = 0; i < 10; i++) {
for (var j = 0; j < 4; j++) {
maze[i][j] = new Array();
}
}
```

Translating the random string into the new data structure

```
n=0;
for (var y = 0; y < 4; y++) {
for (var x = n; x < 10; x+= 2) {
maze[x][y][0] = number[4*x+5*y];
maze[x][y][1] = number[4*x+5*y+1];
maze[x][y][2] = number[4*x+5*y+2];
maze[x][y][3] = number[4*x+5*y+3];
}
count++;
if (count%2 == 1) {n=1}
if (count%2 == 0) {n=0}
}
```

Each entry in the maze structure represents a coordinate, [x][y] and a direction [0,1,2 or 3]. Each entry is a 1 or 0 and represents whether a line is drawn from the coordinate to the point above (direction 0), to the right of it (1), below (2) or to the left (3). This is explained below and an example is given.



The final stage is to use this data structure to sketch the maze onto the screen, and then the canvas is stroked to draw the maze to the screen.

This data structure is not eventually used due to the error shown in the diagram with 2 methods for the same mazes. This could lead to potential data inconsistencies that should be avoided if possible.

Final maze generation algorithm – Stage 4

To generate a random valid maze, a number of steps need to be taken.

- The first is defining all the data structures needed, examples of these are: to store the location of all the points of individual trees in the “trees” array, an array to store whether or not the nth tree generated has crossed another tree, the “Tcross” array, an array to keep track of points that we don’t want to be part of any other tree, done points, array “DP”, and

array to keep track of which trees are walltrees, the “walltrees” array. As well as this a number of other variables are defined.

- The main loop in the program creates the random maze by firstly choosing a random point not in “DP”, then updating whether or not the current tree is a wall tree and adding this first point to our current tree.
- We then choose a random length for our tree and then add this many more points to the tree.
- Each time the points are chosen at random and checked for validity, this is done using the valid function. If we are not able to find a valid point to add to our tree at any point, having checked all 4 surrounding points, we stop growing the tree before the length previously generated, since there is nowhere to grow it too.
- Having produced a tree, this process is repeated until we break out of the loop. The conditions for breaking the loop are that 50 random points were chosen, and all were in the “DP” array, this adds another element of randomness to the maze generation. Once we have broken the loop we then draw the maze to the screen from all the points in the trees array.

The valid function is essential to the success of this algorithm, and it checks the inputted point for a number of key things.

- Firstly, we check that the point is inside the maze, i.e. not at x coordinate -1.
- We then check that, if we are already a wall tree, so a tree with a point on the outer maze wall, we are not now adding a point that is on the wall again.
- We check that we haven’t chosen a point on the outer wall of the maze is we have crossed another tree. If the point that we have chosen is in the “DP” array, we know that another tree has already passed through this point and so we are crossing another tree. The algorithm allows tree crossing to occur, but we have to check a number of thing is this case.
    - Firstly we check that the two trees crossing are not both attached to the outer wall, by looking them up in the wall trees array, this is done using another function, which takes the point we are crossing, finds the tree passing through it and then checking if it is attached to the outer wall.
    - We also check that the current tree has not previously crossed a wall, since we only allow the current tree to pass one other tree. This is not limiting, since a generated tree can still be crossed by multiple trees that have been generated after it, since we only stop the current tree from crossing more than once.
    - If all these conditions are fulfilled, we update the relevant data structures and accept the point as valid.

Pseudo code:

Create data structures

Find random point not in “DP”

Update “walltree” array

Add other random points to tree

Repeat for other trees

Draw trees to screen

Allowing the input of maze solutions – stage 6

The first thing that needs to be added at this stage is a data structure to keep track of which points walls are between. Having added this, we need to add a counter to be moved around the screen. This counter is drawn onto the screen, then we monitor for a key press. To monitor this key press we use the function doKeyDown(). To help explain, the two most important lines are below.

```
function doKeyDown(e) {
```

```
if (e.keyCode == 87) {
```

The first function is run when a key is pressed, and then assigns the integer value of the key to a new variable "e". We then read this variable with ".keyCode", which allows us to run different code for different keys. In the example shows, 87 stands for the "w" key. This is the case because 87 is the ASCII code for a 'w', and keycodes are based on the ASCII table.

In the event of a key press, we check that the move is possible; looking at the data structure we just created.

```
if ((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) /
(canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))][((x +
(canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))] == 1)) {
    //alert("invaild move");
  } else {
```

This is the code to check that the move is possible; as the code shows there are 2 things that the checked. The first thing that would make the move invalid is if we were in the top row of cells in the maze. The "y" variable represents the pixel coordinates of the counter and "canvas.height" is the total height of the canvas in pixels, from the top of the code we find that this is 400 pixels. When the counter is in the top row of the maze, its value is equal to an eighth of the total height, if this is the case we know that the move is not valid, as the counter will move off the maze. The second condition is much longer and can be seen starting after the "II", the or symbol. In order to check to see if a wall is blocking our path we read in two pairs of coordinates, and a 1 is returned if a wall is present. But to get the coordinates, we must convert our pixel coordinates into array positions, an example being that a wall blocking an upwards move from pixel coordinates (250, 250) would run from coordinates (2,2) to (3,2). These conversions done as shown in the second condition of the if statement.

We then clear the old counter from the screen and draw a new counter on the cell that we have moved too. In order to display information about the move numbers and number of times visited we need a "visited" array and a "NextPos" array, in order to know what to write into the cell we have just entered when we move there. .fillText() is used to draw this information to the screen.

```
context.fillText(MoveCounter, x - 6, y + 14);
```

This line draws the value of the variable MoveCounter to the coordinates ((x - 6), (y+14)).

Pseudo code:

Draw counter and initial cell info to screen

Wait for keypress

Validate keypress

Move counter

Add new info to next cell

Making the path more clear and maze resetting – stage 7

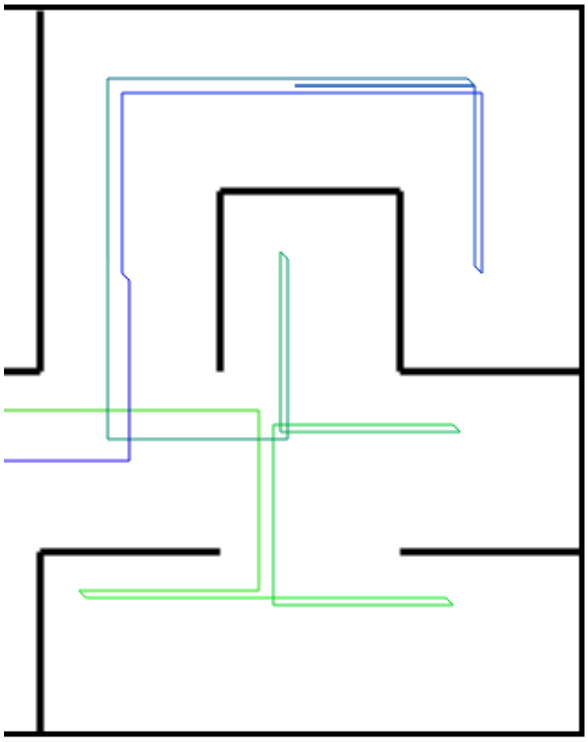
The next stage of the project was to make the path taken by the user clearer. This is important for users trying to retrace their previous saves in order to compare them to new solutions. Although this is possible currently, I believe that the system could be improved to a continuous path in place of a string of numbers. Because the current maze being displayed is displaying a lot of useful information for the user, another identical maze is added below in order to show the path taken on a clear maze. Another step to make the path clearer is to continually change the colour, so that when the path crosses itself, it is still easy to follow. Below a section of the lower maze is shown, with the path taken by the user. Shifting also occurs when the cell we are moving into has been visited more times than the cell we are currently in, this shift stops the path from being drawn over itself. This “shift” is the line representing the path moving from its current position to move down and to the right to stay in the same cell, but now when it moves off, it will not overlap with other paths being drawn. The aim of shifting is to stop overlapping occurring.

```
function DrawToSecond(NX, NY) {

    if ((visited[((NX - (canvas.width / 20)) / (canvas.width / 10))][((NY - (canvas.heig
        shiftTemp = shift;
        //dont want a massive shift!
        if (shift < 81) {
            shift += 4;
        }
        context2.beginPath();
        context2.moveTo((CurrentX + shiftTemp), (CurrentY + shiftTemp));
        context2.lineTo((CurrentX + shift), (CurrentY + shift));
        //change line colour
        if ((redVal > 0) && (blueVal == 0)) {
            redVal += (-10);
            greenVal += 10;
        } else if (greenVal > 0) {
            greenVal += (-10);
            blueVal += 10;
        } else if (blueVal > 0) {
            blueVal += (-10);
            redVal += 10;
        }
        context2.strokeStyle = "rgb(" + redVal + "," + greenVal + "," + blueVal + ")";
        context2.stroke();
    }
    context2.beginPath();
    context2.moveTo((CurrentX + shift), (CurrentY + shift));
    context2.lineTo((NX + shift - 40), (NY + shift - 40));
    //change line colour
    if ((redVal > 0) && (blueVal == 0)) {
        redVal += (-10);
        greenVal += 10;
    } else if (greenVal > 0) {
        greenVal += (-10);
        blueVal += 10;
    } else if (blueVal > 0) {
        blueVal += (-10);
        redVal += 10;
    }
    context2.strokeStyle = "rgb(" + redVal + "," + greenVal + "," + blueVal + ")";
    context2.stroke();

    CurrentX = NX - 40;
    CurrentY = NY - 40;
```

This is the function drawing the line described above to the second maze. The first stage is to increment the shift by 4 pixels, under the conditions that we have already been to the new cell and that the current shift value is below 81, so we have made 20 shifts or less. This is done to stop shift going too far, the result of which would be shifting the line out of the current cell. If we have made a shift we alter colour and then make the shift from the old position to the new position within the same cell. This movement is always 4 pixels down and right, as is visible from the picture of the shift occurring. We then exit the if statement, shift again and move the line to the new cell, keeping track of the last cell we were in with the “CurrentX” and “CurrentY” variables, so we can draw the line to the next inputted point next time. Coordinates are inputted into the function as parameters, “NX” and “NY”. A double change in colour occurs in the case that a shift is needed, this is done to change

the colour more quickly, as shifting occurs in areas that have more lines in, this makes it even more important to ensure that colours are quickly changing. This more rapid change in colour helps to improve the visibility of the line, when the path becomes more complicated with additional lines.

To alter the colour of the line, we start the line being red, then reduce the red value and increase the green until the line is green. This is done from green to blue and then back from blue to red. This process is looped and results in a line continuously changing colour. Having altered the variables, we draw the new section of line to the screen by inputting the variables representing the colour, “greenVal”, “blueVal” and “redVal” into the stroke procedure. This draws the new section of line with the altered colours.



The other feature added at this stage was a reset button. This clears the maze and all data structures and then redraws the maze from the points saved in the trees array. This is a challenge because when the maze is initially drawn, it is drawn at the same time that is it generated and we need to recreate this by keeping track of how many trees were produced and simulate the process again using a for loop.

Saving and loading – stage 11

To allow users to keep previous solutions, a save feature is needed. In stage 10, a feature to save and load a single maze was added, using a function to save and load cookies. These functions are used in “SaveMaze” and “LoadMaze” functions, linked to corresponding buttons on the page. These functions store separate cookies for initial coordinates, which is important since at this stage starting point is random, the “currenttree” variable to keep track of the number of trees we have. Because the information about how many times we have been to a cell, etc, are not stored in arrays and storing this information would require a number of new data structures, path taken by the user is saved into an array as a string of moves and reproduced later. We need to add new data structures

to keep track of the moves we have made, so at this point we are adding moves into these arrays every time the user makes a move. These arrays are joined to a string and stored in a single cookie, as well as a variable to store the number of moves in the array. Finally, the "trees" array and "structure" array are stored through joining, however the "structure" array can only be stored as a string of "1"s and "0"s with no spaces, since a cookie is limited to 4096 bytes. This information is then reloaded using the "LoadMaze" function. In stage 11 this feature is expanded, with cookies being stored with names taken from user input, in a text box. These stored items are then available for the user to select from a dropdown list.

Random solve – stage 13

At this stage, I will add a feature to allow the user to randomly solve the maze. The algorithm will work by randomly choosing a direction that is open and hasn't previously been visited. If one is not available, then a move is popped off the stack, this is a list of moves that have been made. This means that the algorithm will backtrack the path that it has taken until a move is available. This algorithm is shown below.

Pseudo code:

> Choose random direction that hasn't been visited and doesn't have a wall blocking that direction.
>
> Move the counter to the new square, by clearing and redrawing. Add the move to the stack so that the path taken can be tracked
>
> If no move is available, pop the last move off the stack and move in the opposite direction to avoid getting stuck.
>
> When no move is available and the stack is empty, we know that the whole maze has been cover and we are back to the start of the maze.

This algorithm can be used by the user to provide a random path around the maze using a simple algorithm. This will allow users to test their algorithms against a simple, know algorithm. One of the first stages of algorithm testing could then be that the proposed algorithm is more efficient that this inbuilt inefficient one at least most of the time as an initial indicator of algorithm efficiency.

This function will then be linked to a button on the page so that users can access it.

Follow wall – stage 16

In this final stage, the second inbuilt procedure is added. This procedure will allow the user to simulate the counter following the wall next to it. It works by using 3 new functions that have been added to the system. They are "FrontOpen", "LeftOpen" and "MoveDir". These are explained below:

FrontOpen : Returns true if the wall in front of the robot is open, else returns false. This function reads in direction, and in a case statement, works out the direction that would be forward for the robot. It then checks to see if this is open.

LeftOpen : Returns true if the wall to the left of the robot is open, else returns false. Works in the same way as the FrontOpen function.

MoveDir: Move the Robot forward, in the direction given. This function also uses a case statement to work out the correct direction, then move the counter.

The function itself works by using the functions shown above. In order to follow the wall the following algorithm is used:

Pseudo code:

```
Set initial direction (North)
Record initial coordinates
If (LeftOpen = true) turn left
Else if (FrontOpen = true) go forwards
Else turn right
While (Initial cords != current cords)
    If (LeftOpen = true) turn left
    Else if (FrontOpen = true) go forwards
    Else turn right
```

The first move is before the while loop so that the loop is not immediately exited. This pseudo code describes how the code for the follow wall procedure works.

Technical Solution

```
<!DOCTYPE html>
<html>

<head>
   <title>Page title
   </title>
   <style type="text/css">
      canvas {
         border-style: solid;
                        font-family:"Verdana", Times, serif;
                        background: -moz-linear-gradient(top,#D0D0D0, #FFFFFF );
                        align: center;
                        box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;


      }
                body {
                font-family:"Verdana", Times, serif;
                background-image: -moz-radial-gradient(center, circle closest-corner, #FFFFFF 0%, #2b4dff 180%);
                }
                button{
                font-family:"Verdana", Times, serif;
                background: -moz-linear-gradient(top,#FFFFFF, #2b4dff );
                align: center;
                border-radius: 55px 55px 25px 25px;
                box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;
                }
                button:hover{
                font-family:"Verdana", Times, serif;
                background: -moz-linear-gradient(top,#2b4dff,#FFFFFF );
                align: center;
                border-radius: 55px 55px 25px 25px;
                box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;

                }
                select{
                font-family:"Verdana", Times, serif;
```

```
            background: -moz-linear-gradient(top,#FFFFFF, #2b4dff );
            align: center;
            box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;

            }
            input{
            font-family:"Verdana", Times, serif;
            background: -moz-linear-gradient(top,#FFFFFF, #2b4dff );
            align: center;
            box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;

            }
            input:hover{
            font-family:"Verdana", Times, serif;
            background: -moz-linear-gradient(top,#2b4dff,#FFFFFF);
            align: center;
            box-shadow: inset 0px 0px 0px #3e9cbf, 0px 1px 0px 0px #205c73, 0px 10px 5px #999;

            }


   </style>

</head>

<body>
   <h3> The MazeSolving Page</h3>

<center><table border="0" width="1000"
  <tr>
  <center>

   <td><canvas id="b" width="1000" height="400" tabindex="0"></canvas></td>
</center>
  </tr>
  <tr>
<center>
```

```
   <td><button onclick="RedrawMaze()">Reset Maze.</button><button onclick="SaveMaze()">Save</button><button
onclick="LoadMaze()">Load</button><input type="text" name="txt" id="txt" />
       <select id="mySelect">

       </select>
   </td>
</center>
  </tr>
  <tr>
  <center>

   <td><button onclick="RandomSolve()">RandomSolve</button><button onclick="FloatOnOff()">Turn Floating walls on/off</button><button
onclick="FollowWall()">FollowWall</button></td>
</center>
  </tr>
   <tr>
   <center>

   <td>    <p>Press tab to begin. Use W, A, S and D to move.</p>
   <p>Below is the path taken.</p></td>
</center>
  </tr>
   <tr>
<center>

   <td>    <canvas id="c" width="1000" height="400" tabindex="0"></canvas></td>
</center>
  </tr>
</table></center>

   <p>Rhys Patten</p>

   <script>
      var canvas = document.getElementById("b");
      var context = canvas.getContext('2d');
      var canvas2 = document.getElementById("c"); //second canvas here!
      var context2 = canvas2.getContext('2d');
```

```
function readCookie(name) {
    var nameEQ = name + "=";
    var ca = document.cookie.split(';');
    for (var i = 0; i < ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' ') c = c.substring(1, c.length);
        if (c.indexOf(nameEQ) == 0) return c.substring(nameEQ.length, c.length);
    }
    //return null;
}

if (readCookie('FLOATING') == 'true') {
    var FLOATING = true;
} else if (readCookie('FLOATING') == 'false') {
    var FLOATING = false;
} else {
    setCookie('FLOATING', 'true', 10);
    FLOATING = true;
}
 //var FLOATING = readCookie('FLOATING');

 // NOTE THAT THIS MAKES THE MAZE A FLOATING WALL MAZE (TRUE = FLOATING, FALSE = WALL TREES ONLY)

var countx = 0;
var county = 0;
var count = 0;
var maze = new Array();

var Tcross = new Array();

var trees = new Array();

var walltrees = new Array();

var structure = new Array(); // for checking if lines exist!

for (var x1 = 0; x1 < 11; x1++) {
    structure[x1] = new Array();
```

```
}

for (var x1 = 0; x1 < 11; x1++) {
    for (var y1 = 0; y1 < 5; y1++) {
        structure[x1][y1] = new Array();
    }
}

for (var x1 = 0; x1 < 11; x1++) {
    for (var y1 = 0; y1 < 5; y1++) {
        for (var x2 = 0; x2 < 11; x2++) {
            structure[x1][y1][x2] = new Array();
        }
    }
}

for (var x1 = 0; x1 < 11; x1++) {
    for (var y1 = 0; y1 < 5; y1++) {
        for (var x2 = 0; x2 < 11; x2++) {
            for (var y2 = 0; y2 < 5; y2++) {
                structure[x1][y1][x2][y2] = 0;
            }
        }
    }
}

for (var i = 0; i < 50; i++) {
    trees[i] = new Array();
}
for (var i = 0; i < 50; i++) {
    for (var j = 0; j < 6; j++) {
        trees[i][j] = new Array();
    }
}
for (var i = 0; i < 50; i++) {
    for (var j = 0; j < 6; j++) {
        trees[i][j][0] = 0;
        trees[i][j][1] = 0;
```

```
    }
}

 //

for (var i = 0; i < 100; i++) {
    Tcross[i] = 0;
}

var DP = new Array(); // points that have a wall at them
for (var i = 0; i < 11; i++) {
    DP[i] = new Array();
}

for (var i = 0; i < 11; i++) {
    for (var j = 0; j < 5; j++) {
        DP[i][j] = new Array();
    }
}

var corner = false;
var edge = false;
var walls = 0;
var XorY = 0;
var Xpos = 0;
var Ypos = 0;
var PointX = 0;
var PointY = 0;
var wallno = 0;
var crossed = 0;
var ExitCount = 0;

var currenttree = 0; // this tells us which tree we are growing!

DP[0][0] = 1;
DP[10][0] = 1;
DP[0][4] = 1;
DP[10][4] = 1; //avoid corners!
```

```
function IsOnWallTree(X, Y) {

    var result = false;

    for (var t = 0; t < currenttree; t++) {
        for (var w = 0; w < 6; w++) {
            if ((trees[t][w][0] == X) && (trees[t][w][1] == Y)) {
                if (walltrees[t] == true) {
                    result = true;
                }
            }
        }
    }

    return result;


}

function valid(X, Y) {

    var result = true;


    if ((X > -1) && (Y > -1) && (X < 11) && (Y < 5)) {

        if (FLOATING == false) {
            if (walltrees[currenttree] == false) {
                if ((X == 0) || (X == 10) || (Y == 0) || (Y == 4)) { //do nothing
                } else {
                    result = false;
                }
            }
        }

        for (var i = 0; i < wallno; i++) {
            if ((trees[currenttree][i][0] == X) && (trees[currenttree][i][1] == Y)) {
```

```
        result = false; // 100% this is a FAIL EVERY TIME! WE CANNOT CROSS OUR OWN PATH
    }
}

if ((walltrees[currenttree] == true) && ((X == 10) || (Y == 4) || (X == 0) || (Y == 0))) { //we are hitting the wall twice!

    result = false;

}

if (((X == 0) || (Y == 0) || (X == 10) || (Y == 4)) && (Tcross[currenttree] == 1)) {
    result = false;
}


if (DP[X][Y] == 1) {
    //we have crossed a done point!
    if ((IsOnWallTree(X, Y) === true) && (walltrees[currenttree])) {
        //TWO WALLTREES ARE CROSSING, FAIL
        result = false;
    }
    if (Tcross[currenttree] == 1) {
        result = false;
    }
    if ((result == true) && (IsOnWallTree(X, Y) === true)) {
        // make current tree a walltree as it is crossing a walltree
        walltrees[currenttree] = true;
    }
    if (result == true) {
        Tcross[currenttree] = 1;
    }

}
if (((X == 0) || (Y == 0) || (X == 10) || (Y == 4)) && (Tcross[currenttree] == 1)) {
    result = false;
}
```

```
            return result;
        } else {
            return false;
        }
    }

    var finished = false;

    while (finished === false) {
        PointX = 0;
        PointY = 0;
        NewX = 0;
        NewY = 0;
        var nextpoint = false;
        ExitCount = 0;
        while (nextpoint == false) {
            NewX = Math.floor(Math.random() * 11);
            NewY = Math.floor(Math.random() * 5);
            if (FLOATING == true) {
                if (DP[NewX][NewY] == 0) {
                    nextpoint = true;
                }
            }
            if (FLOATING == false) {
                if ((DP[NewX][NewY] == 0) && ((NewX == 10) || (NewX == 0) || (NewY == 4) || (NewY == 0))) {
                    nextpoint = true;
                }

            }
            ExitCount++;

            if (ExitCount > 50) {

                finished = true;
                nextpoint = true; //we looked 50 times for points, could not find one, so the maze is probably pretty much filled. This is ok as it adds more randomness! i.e. the maze isn't filled every time!
            }
        }
```

```
if (finished == false) {
    DP[NewX][NewY] = 1;

    if ((NewX == 10) || (NewY == 4) || (NewX == 0) || (NewY == 0)) {
        walltrees[currenttree] = true;
    }

    walls = Math.floor((Math.random() * 5) + 2); // cannot have 0 walls!

    trees[currenttree][0][0] = NewX; // add it to the current tree
    trees[currenttree][0][1] = NewY;



    PointX = NewX;
    PointY = NewY;

    for (wallno = 1; wallno < walls; wallno++) {

        Xpos = 0;
        Ypos = 0;

        //need to find the next point to add to tree

        XorY = Math.floor(Math.random() * 2); // random direction


        if (XorY == 1) { //We move in X direction
            Xpos = Math.floor(Math.random() * 2) + 1;
            if (Xpos == 1) { // X+1
                PointX++;
            } else {
                PointX--;
            }
        } else {
            Ypos = Math.floor(Math.random() * 2) + 1;

            if (Ypos == 1) { // X+1
```

```
            PointY++;
        } else {

            PointY--;
        }
    }

    if (valid(PointX, PointY) == true) {

        count = 10; // ie we have the point!
    } else {

        count = 0;
        while (count < 3) {

            if (Xpos == 1) {
                Xpos = 0;
                Ypos = 1; // so that next time we loop we hit a different if statement!
                PointX--;
                PointY++;
            } else if (Xpos == 2) {
                Xpos = 0;
                Ypos = 2;
                PointX++;
                PointY--;
            } else if (Ypos == 1) {
                Xpos = 2;
                Ypos = 0;
                PointY--;
                PointX--;
            } else if (Ypos == 2) {
                Xpos = 1;
                Ypos = 0;
                PointY++;
                PointX++;
            }
            if (valid(PointX, PointY) == true) {
```

```
                    count = 10; // so that we exit AND know that we succeeded in finding a valid point
                } else {
                    count++;
                }

            }
        }
        // if count is 10 we add the point to the tree, else it has failed and we exit
        if ((count == 10) && (valid(PointX, PointY) == true)) {

            trees[currenttree][wallno][0] = PointX; // add it to the current tree
            trees[currenttree][wallno][1] = PointY;

            if ((PointX == 10) || (PointY == 4) || (PointX == 0) || (PointY == 0)) {
                walltrees[currenttree] = true;
            }

            DP[PointX][PointY] = 1; //we have done this point! this must come after the valid check!
        } else {

        }

    }

    var acceptable = false;

    if (FLOATING == false) {

        if ((((((trees[currenttree][0][0] - trees[currenttree][1][0]) == 1) || (trees[currenttree][0][0] - trees[currenttree][1][0]) == -1)) &&
(trees[currenttree][0][1] == trees[currenttree][1][1])) || (((((trees[currenttree][0][1] - trees[currenttree][1][1]) == 1) || (trees[currenttree][0][1] -
trees[currenttree][1][1]) == -1)) && (trees[currenttree][0][0] == trees[currenttree][1][0]))) {
            if ((trees[currenttree][0][0] == 0) || (trees[currenttree][0][0] == 10) || (trees[currenttree][0][1] == 0) || (trees[currenttree][0][1] ==
4)) {
                if ((trees[currenttree][1][0] == 0) || (trees[currenttree][1][0] == 10) || (trees[currenttree][1][1] == 0) || (trees[currenttree][1][1]
== 4)) {} else {
                    acceptable = true;
                }
            }
```

```
                }
            }
            if (FLOATING == true) {
                acceptable = true;
            }

            if (acceptable == true) {

                for (var j = 1; j < 6; j++) {
                    if (((trees[currenttree][j][0] == 0) && (trees[currenttree][j][1] == 0)) || ((trees[currenttree][j - 1][0] == 0) && (trees[currenttree][j -
1][1] == 0))) {
                        //fail we do not draw
                    } else {
                        context.moveTo(((trees[currenttree][j - 1][0] * (canvas.width / 10)) + 0.5), ((trees[currenttree][j - 1][1] * (canvas.height / 4)) +
0.5));
                        context.lineTo(((trees[currenttree][j][0] * (canvas.width / 10)) + 0.5), ((trees[currenttree][j][1] * (canvas.height / 4)) + 0.5));

                        context2.moveTo(((trees[currenttree][j - 1][0] * (canvas2.width / 10)) + 0.5), ((trees[currenttree][j - 1][1] * (canvas2.height / 4))
+ 0.5));
                        context2.lineTo(((trees[currenttree][j][0] * (canvas2.width / 10)) + 0.5), ((trees[currenttree][j][1] * (canvas2.height / 4)) + 0.5));

                        structure[trees[currenttree][j - 1][0]][trees[currenttree][j - 1][1]][trees[currenttree][j][0]][trees[currenttree][j][1]] = 1;
                        structure[trees[currenttree][j][0]][trees[currenttree][j][1]][trees[currenttree][j - 1][0]][trees[currenttree][j - 1][1]] = 1; //+ the
inverse!
                    }
                }
            }
            currenttree++;
        }
    }


    function setCookie(c_name, value, exdays) {
        var exdate = new Date();
        exdate.setDate(exdate.getDate() + exdays);
        var c_value = escape(value) + ((exdays == null) ? "" : "; expires=" + exdate.toUTCString());
        document.cookie = c_name + "=" + c_value;
    }
```

```
function eraseCookie(name) {
    setCookie(name, "", -1);
}

var MovesMade = new Array();
var MovesMadeCount = 0;

var SaveCounter = 0;

while ((readCookie(SaveCounter + "Save"))) {
    var x = document.getElementById("mySelect");
    var option = document.createElement("option");
    option.text = readCookie(SaveCounter + "Save");
    try {
        x.add(option, x.options[null]);
    } catch (e) {
        x.add(option, null);
    }
    SaveCounter++;
}

function FloatOnOff() {

    if (FLOATING == true) {
        setCookie('FLOATING', 'false', 10);
        FLOATING = false;
        alert('Floating walls OFF');
    } else {
        setCookie('FLOATING', 'true', 10);
        FLOATING = true;
        alert('Floating walls ON');
    }


}
```

```
function SaveMaze() {
    var txtbox = document.getElementById('txt');
    var value = txtbox.value;
    var x = document.getElementById("mySelect");
    var option = document.createElement("option");
    option.text = value;
    try {
        x.add(option, x.options[null]);
    } catch (e) {
        x.add(option, null);
    }

    setCookie(SaveCounter + "Save", value, 10);

    setCookie(value + "CT", currenttree, 10);
    setCookie(value + "xInitial", xInitial, 10);
    setCookie(value + "yInitial", yInitial, 10);

    var MovesMadeCookie = MovesMade.join();
    setCookie(value + "MovesMadeCookie", MovesMadeCookie, 10);
    setCookie(value + "MovesMadeCount", MovesMadeCount, 10);

    var hello = trees.join();
    setCookie(value + "trees111", hello, 10);

    var hello22 = structure.join("");
    setCookie(value + "Cookie2", hello22.replace(/,/g, ""), 10);

}

function LoadMaze() {

    var e = document.getElementById("mySelect");
    var loadvalue = e.options[e.selectedIndex].text;

    currenttree = readCookie(loadvalue + "CT");

    xInitial = readCookie(loadvalue + "xInitial");
```

```
yInitial = readCookie(loadvalue + "yInitial");

xInitial = parseInt(xInitial, 10);
yInitial = parseInt(yInitial, 10)

var MovesMadeCookie = readCookie(loadvalue + "MovesMadeCookie");

MovesMade = MovesMadeCookie.split("%2C");
MovesMadeCount = readCookie(loadvalue + "MovesMadeCount");
MovesMadeCount = parseInt(MovesMadeCount, 10);

var hello1 = readCookie(loadvalue + "trees111");

var trees1 = hello1.split("%2C");

var hello221 = readCookie(loadvalue + "Cookie2");

var structureTemp = hello221.split("");


var counter = 0;
for (var i = 0; i < 50; i++) {
    for (var j = 0; j < 6; j++) {
        trees[i][j][0] = trees1[counter];
        counter++;
        trees[i][j][1] = trees1[counter];
        counter++;
    }
}

counter = 0;

for (var x1 = 0; x1 < 11; x1++) {
    for (var y1 = 0; y1 < 5; y1++) {
        for (var x2 = 0; x2 < 11; x2++) {
            for (var y2 = 0; y2 < 5; y2++) {
                structure[x1][y1][x2][y2] = structureTemp[counter];
                counter++;
```

```
                }
            }
        }
    }
    RedrawMaze();

    for (var i = 0; i < MovesMadeCount; i++) {
        MovesMade[i] = parseInt(MovesMade[i], 10);

        if (MovesMade[i] == 0) {
            if ((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invaild move");
            } else {

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y - (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
```

```
                DrawToSecond(x, y);

            }
        } else if (MovesMade[i] == 1) {
            if ((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invaild move");
            } else {

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y + (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

            }
        } else if (MovesMade[i] == 2) {
```

```
            if ((x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invalid move");
            } else {

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x - (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

            }
        } else if (MovesMade[i] == 3) {
            if ((x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invalid move");
            } else {
```

```
                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x + (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

            }
        }

    }


}

function RedrawMaze() {

    canvas.width = canvas.width;
    canvas2.width = canvas2.width;
```

```
        for (var tempcurrenttree = 0; tempcurrenttree < currenttree; tempcurrenttree++) {

            var acceptable = false;

            if (FLOATING == false) {

                if ((((((trees[tempcurrenttree][0][0] - trees[tempcurrenttree][1][0]) == 1) || (trees[tempcurrenttree][0][0] -
trees[tempcurrenttree][1][0]) == -1)) && (trees[tempcurrenttree][0][1] == trees[tempcurrenttree][1][1])) || (((((trees[tempcurrenttree][0][1] -
trees[tempcurrenttree][1][1]) == 1) || (trees[tempcurrenttree][0][1] - trees[tempcurrenttree][1][1]) == -1)) && (trees[tempcurrenttree][0][0] ==
trees[tempcurrenttree][1][0]))) {
                    if ((trees[tempcurrenttree][0][0] == 0) || (trees[tempcurrenttree][0][0] == 10) || (trees[tempcurrenttree][0][1] == 0) ||
(trees[tempcurrenttree][0][1] == 4)) {
                        if ((trees[tempcurrenttree][1][0] == 0) || (trees[tempcurrenttree][1][0] == 10) || (trees[tempcurrenttree][1][1] == 0) ||
(trees[tempcurrenttree][1][1] == 4)) {} else {
                            acceptable = true;
                        }
                    }
                }
            }
            if (FLOATING == true) {
                acceptable = true;
            }

            if (acceptable == true) {

                for (var j = 1; j < 6; j++) {
                    if (((trees[tempcurrenttree][j][0] == 0) && (trees[tempcurrenttree][j][1] == 0)) || ((trees[tempcurrenttree][j - 1][0] == 0) &&
(trees[tempcurrenttree][j - 1][1] == 0))) {
                        //fail we do not draw
                    } else {
                        context.moveTo(((trees[tempcurrenttree][j - 1][0] * (canvas.width / 10)) + 0.5), ((trees[tempcurrenttree][j - 1][1] *
(canvas.height / 4)) + 0.5));
                        context.lineTo(((trees[tempcurrenttree][j][0] * (canvas.width / 10)) + 0.5), ((trees[tempcurrenttree][j][1] * (canvas.height / 4)) +
0.5));

                        context2.moveTo(((trees[tempcurrenttree][j - 1][0] * (canvas2.width / 10)) + 0.5), ((trees[tempcurrenttree][j - 1][1] *
(canvas2.height / 4)) + 0.5));
```

```
                    context2.lineTo(((trees[tempcurrenttree][j][0] * (canvas2.width / 10)) + 0.5), ((trees[tempcurrenttree][j][1] * (canvas2.height / 4)) + 0.5));
                }
            }
        }
    }
    context.lineWidth = 4;
    context.strokeStyle = "blue";
    context.stroke();
    context2.lineWidth = 4;
    context2.strokeStyle = "black";
    context2.stroke();
    context2.lineWidth = 1;

    for (var i = 0; i < 11; i++) {
        for (var j = 0; j < 5; j++) {
            visited[i][j] = 0;
        }
    }

    for (var i = 0; i < 11; i++) {
        for (var j = 0; j < 5; j++) {
            NextPos[i][j] = 0;
        }
    }

    x = xInitial;
    y = yInitial;
    radius = 27;
    MoveCounter = 1;

    CurrentX = x - 40;
    CurrentY = y - 40;

    shift = 0;
    shiftTemp = 0;
    redVal = 250;
    greenVal = 0;
```

```
        blueVal = 0;

        canvas_context = canvas.getContext("2d");
        context.beginPath();
        context.arc(x, y, 30, 0, 2 * Math.PI, false);
        context.fill();
        context.arc(x, y, 27, 0, 2 * Math.PI, false);
        context.fill();

        visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;

        context.fillStyle = "white";
        context.font = "bold 16px Arial";
        context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y - 4);
        context.fillText(MoveCounter, x - 6, y + 14);

        AddNextMove();

    }
    context.lineWidth = 4;
    context.strokeStyle = "blue";
    context.stroke();
    context2.lineWidth = 4;
    context2.strokeStyle = "black";
    context2.stroke();
    context2.lineWidth = 1;
    var canvas = document.getElementById("b");
    canvas.addEventListener('keydown', doKeyDown, true);

    var visited = new Array();

    for (var i = 0; i < 11; i++) {
        visited[i] = new Array();
    }

    for (var i = 0; i < 11; i++) {
        for (var j = 0; j < 5; j++) {
            visited[i][j] = 0;
```

```
    }
}

var NextPos = new Array(); // for where we are placing the next numbers

for (var i = 0; i < 11; i++) {
    NextPos[i] = new Array();
}

for (var i = 0; i < 11; i++) {
    for (var j = 0; j < 5; j++) {
        NextPos[i][j] = 0;
    }
}
var xInitial = Math.floor(Math.random() * 10) * (canvas.width / 10) + (canvas.width / 20);
var yInitial = Math.floor(Math.random() * 4) * (canvas.height / 4) + (canvas.height / 8);
var x = xInitial;
var y = yInitial;
var radius = 27;
var MoveCounter = 1;

var CurrentX = x - 40;
var CurrentY = y - 40;
var shift = 0;
var shiftTemp = 0;
var redVal = 250;
var greenVal = 0;
var blueVal = 0;

canvas_context = canvas.getContext("2d");
context.beginPath();
context.arc(x, y, 30, 0, 2 * Math.PI, false);
context.fill();
context.arc(x, y, 27, 0, 2 * Math.PI, false);
context.fill();


visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
```

```
        context.fillStyle = "white";
        context.font = "bold 16px Arial";
        context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y - 4);
        context.fillText(MoveCounter, x - 6, y + 14);

        function DrawToSecond(NX, NY) {

            if ((visited[((NX - (canvas.width / 20)) / (canvas.width / 10))][((NY - (canvas.height / 8)) / (canvas.height / 4))]) > ((visited[((((CurrentX +
40) - (canvas.width / 20)) / (canvas.width / 10))][((((CurrentY + 40) - (canvas.height / 8)) / (canvas.height / 4))])))) {
                shiftTemp = shift;
                //dont want a massive shift!
                if (shift < 81) {
                    shift += 4;
                }
                context2.beginPath();
                context2.moveTo((CurrentX + shiftTemp), (CurrentY + shiftTemp));
                context2.lineTo((CurrentX + shift), (CurrentY + shift));
                //change line colour
                if ((redVal > 0) && (blueVal == 0)) {
                    redVal += (-10);
                    greenVal += 10;
                } else if (greenVal > 0) {
                    greenVal += (-10);
                    blueVal += 10;
                } else if (blueVal > 0) {
                    blueVal += (-10);
                    redVal += 10;
                }
                context2.strokeStyle = "rgb(" + redVal + "," + greenVal + "," + blueVal + ")";
                context2.stroke();
            }
            context2.beginPath();
            context2.moveTo((CurrentX + shift), (CurrentY + shift));
            context2.lineTo((NX + shift - 40), (NY + shift - 40));
            //change line colour
            if ((redVal > 0) && (blueVal == 0)) {
                redVal += (-10);
```

```
            greenVal += 10;
        } else if (greenVal > 0) {
            greenVal += (-10);
            blueVal += 10;
        } else if (blueVal > 0) {
            blueVal += (-10);
            redVal += 10;
        }
        context2.strokeStyle = "rgb(" + redVal + "," + greenVal + "," + blueVal + ")";
        context2.stroke();

        CurrentX = NX - 40;
        CurrentY = NY - 40;


    }

     //start
    function AddNextMove() {
        context.fillStyle = "black";
        context.font = "bold 12px Arial";
        if ((NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]) < 65) {
            if (visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1) {
                context.fillText((MoveCounter), x - 48 + (NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))]), y - 40);
                NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 14;
                if (MoveCounter > 9) {
                    NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
                }
                if (MoveCounter > 99) {
                    NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
                }
            } else {
                context.fillText(("," + MoveCounter), x - 48 + (NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))]), y - 40);
                NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 14;
                if (MoveCounter > 9) {
                    NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
```

```
                }
                if (MoveCounter > 99) {
                    NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
                }
            }
        } else {

            context.fillText((",," + MoveCounter), x - 48 - 65 + (NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))]), y + 40);
            NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 14;
            if (MoveCounter > 9) {
                NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
            }
            if (MoveCounter > 99) {
                NextPos[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] += 7;
            }

        }
    }
     //end

    AddNextMove();

    function doKeyDown(e) {

        //====================
        //     THE W KEY
        //====================
        if (e.keyCode == 87) {
            if ((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invaild move");
            } else {
                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
```

```
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y - (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 0;
                MovesMadeCount++;
            }
        }

        //====================
        //     THE S KEY
        //====================
        if (e.keyCode == 83) {
            if ((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invaild move");
            } else {
                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
```

```
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y + (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 1;
                MovesMadeCount++;
            }
        }

        //====================
        //     THE A KEY
        //====================
        if (e.keyCode == 65) {
            if ((x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invalid move");
            } else {
                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
```

```
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x - (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 2;
                MovesMadeCount++;
            }
        }

        //====================
        //    THE D KEY
        //====================
        if (e.keyCode == 68) {
            if ((x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //alert("invalid move");
            } else {
                context.save();
                context.globalCompositeOperation = 'destination-out';
```

```
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x + (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 3;
                MovesMadeCount++;
            }
        }

    }

                function LeftOpen(dir) {

                //if dir = 1 then left = west, (x-1)
                //if dir = 2 then left = north, (y-1)
                //if dir = 3 then left = east, (x+1)
                //if dir = 4 then left = south, (y+1)
```

```
                switch(dir){

                        case 1:{
                        if ((!((x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height
/ 8)) / (canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
                        return true;
                        }
                        else{
                        return false;
                        }
                        }
                        break;
                        case 2:{
                        if ((!((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height
/ 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
                        return true;
                        }
                        else{
                        return false;
                        }
                        }
                        break;
                        case 3:{
                        if ((!((x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y +
(canvas.height / 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] ==
1))) == 1){
                        return true;
                        }
                        else{
                        return false;
                        }
                        }
                        break;
                        case 4:{
                        if ((!((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height
/ 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
                        return true;
                        }
```

```
                    else{
                    return false;
                    }
                    }
                    break;

                }

            }

            function FrontOpen(dir) {

                switch(dir){

                    case 2:{
                    if ((!((x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y +
(canvas.height / 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] ==
1))) == 1){
                    return true;
                    }
                    else{
                    return false;
                    }
                    }
                    break;
                    case 1:{
                    if ((!((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height
/ 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
                    return true;
                    }
                    else{
                    return false;
                    }
                    }
                    break;
                    case 4:{
                    if ((!((x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height
/ 8)) / (canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
```

```
                                return true;
                                }
                                else{
                                return false;
                                }
                                }
                                break;
                                case 3:{
                                if ((!((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height
/ 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) == 1){
                                return true;
                                }
                                else{
                                return false;
                                }
                                }
                                break;

                        }

                }

                function MoveDir(dir){

                        switch(dir){

                                case 1:{

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
```

```
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y - (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 0;
                MovesMadeCount++;

                                }
                                break;
                                case 2:{

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x + (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
```

```
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 3;
                MovesMadeCount++;

                                }
                                break;
                                case 3:{

                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
                context.fillText(MoveCounter, x - 6, y + 14);
                y = y + (canvas.height / 4);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
```

```
            context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
            context.fillText(MoveCounter, x - 6, y + 14);

            DrawToSecond(x, y);

            MovesMade[MovesMadeCount] = 1;
            MovesMadeCount++;

                        }
                        break;
                        case 4:{

            context.save();
            context.globalCompositeOperation = 'destination-out';
            context.beginPath();
            context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
            context.fill();
            context.restore();
            context.fillStyle = "black";
            context.font = "bold 16px Arial";
            context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
            context.fillText(MoveCounter, x - 6, y + 14);
            x = x - (canvas.width / 10);
            context.beginPath();
            context.arc(x, y, radius, 0, 2 * Math.PI, false);
            context.fill();
            MoveCounter++;
            visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
            AddNextMove();
            context.fillStyle = "white";
            context.font = "bold 16px Arial";
            context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y -
4);
            context.fillText(MoveCounter, x - 6, y + 14);

            DrawToSecond(x, y);
```

```
            MovesMade[MovesMadeCount] = 2;
            MovesMadeCount++;

                        }
                        break;

                }


            }

    function FollowWall() {

            var direction = 1;

            var xStart = ((x + (canvas.width / 20)) / (canvas.width / 10));
            var yStart = ((y + (canvas.height / 8)) / (canvas.height / 4));

            if (LeftOpen(direction) == 1){
            direction--;
            if (direction == 0){ direction = 4;}
            MoveDir(direction);

            }
            else if(FrontOpen(direction) == 1){
            MoveDir(direction);
            }
            else{
            direction++;
            if (direction == 5){ direction = 1;}
            }


            while (!((((x + (canvas.width / 20)) / (canvas.width / 10)) == xStart) && (((y + (canvas.height / 8)) / (canvas.height / 4)) == yStart)
&& (MoveCounter > 40))){

            if (LeftOpen(direction) == 1){
```

```
            direction--;
            if (direction == 0){ direction = 4;}
            MoveDir(direction);

            }
            else if(FrontOpen(direction) == 1){
            MoveDir(direction);
            }
            else{
            direction++;
            if (direction == 5){ direction = 1;}
            }


            }

            }

function RandomSolve() {
    var NewRand = Math.floor(Math.random() * 4);
    var nextmove = 5;
    var lastmove = 0;
    var countvar = 0;
    var MovementNo = 0;
    var Xcoord = ((x - (canvas.width / 20)) / (canvas.width / 10));
    var Ycoord = ((y - (canvas.height / 8)) / (canvas.height / 4));
    var stack = new Array();

    while (MovementNo < 120) {

        NewRand = Math.floor(Math.random() * 4);
        nextmove = 5;
        countvar = 0;
        Xcoord = ((x - (canvas.width / 20)) / (canvas.width / 10));
        Ycoord = ((y - (canvas.height / 8)) / (canvas.height / 4));

        while (nextmove == 5) {
            switch (NewRand) {
```

```
                case 0:
                    {
                        if ((!((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1))) &&
(visited[Xcoord][Ycoord - 1] == 0)) {
                            stack.push(1);
                            nextmove = 0;
                        } else {
                            if (countvar > 3) {
                                countvar = 0;
                                nextmove = stack.pop();
                            }
                            countvar++;
                        }

                    }
                    break;
                case 1:
                    {
                        if ((!((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) &&
(visited[Xcoord][Ycoord + 1] == 0)) {
                            stack.push(0);
                            nextmove = 1;
                        } else {
                            if (countvar > 3) {
                                countvar = 0;
                                nextmove = stack.pop();
                            }
                            countvar++;
                        }
                    }
                    break;
                case 2:
                    {
```

```
                    if ((!(x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1))) && (visited[Xcoord - 1][Ycoord] == 0)) {
                        stack.push(3);
                        nextmove = 2;
                    } else {
                        if (countvar > 3) {
                            countvar = 0;
                            nextmove = stack.pop();
                        }
                        countvar++;
                    }
                }
                break;
            case 3:
                {
                    if ((!(x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1))) && (visited[Xcoord + 1][Ycoord] == 0)) {
                        stack.push(2);
                        nextmove = 3;
                    } else {
                        if (countvar > 3) {
                            countvar = 0;
                            nextmove = stack.pop();
                        }
                        countvar++;
                    }
                }
                break;

            }
            NewRand++;
            if (NewRand > 3) {
                NewRand = 0;
            };
        }
```

```
            //got the move

            if (nextmove == 0) {
                if ((y == (canvas.height * (1 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                    //stack.pop();
                } else {

                    context.save();
                    context.globalCompositeOperation = 'destination-out';
                    context.beginPath();
                    context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                    context.fill();
                    context.restore();
                    context.fillStyle = "black";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);
                    y = y - (canvas.height / 4);
                    context.beginPath();
                    context.arc(x, y, radius, 0, 2 * Math.PI, false);
                    context.fill();
                    MoveCounter++;
                    visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                    AddNextMove();
                    context.fillStyle = "white";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);

                    DrawToSecond(x, y);

                    MovesMade[MovesMadeCount] = 0;
                    MovesMadeCount++;
                }
            }
```

```
            if (nextmove == 1) {
                if ((y == (canvas.height * (7 / 8))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                    //stack.pop();
                } else {

                    context.save();
                    context.globalCompositeOperation = 'destination-out';
                    context.beginPath();
                    context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                    context.fill();
                    context.restore();
                    context.fillStyle = "black";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);
                    y = y + (canvas.height / 4);
                    context.beginPath();
                    context.arc(x, y, radius, 0, 2 * Math.PI, false);
                    context.fill();
                    MoveCounter++;
                    visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                    AddNextMove();
                    context.fillStyle = "white";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);

                    DrawToSecond(x, y);

                    MovesMade[MovesMadeCount] = 1;
                    MovesMadeCount++;
                }
            }
```

```
            if (nextmove == 2) {
                if ((x == (canvas.width * (1 / 20))) || (structure[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) /
(canvas.height / 4))][((x - (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                    //stack.pop();
                } else {
                    context.save();
                    context.globalCompositeOperation = 'destination-out';
                    context.beginPath();
                    context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                    context.fill();
                    context.restore();
                    context.fillStyle = "black";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);
                    x = x - (canvas.width / 10);
                    context.beginPath();
                    context.arc(x, y, radius, 0, 2 * Math.PI, false);
                    context.fill();
                    MoveCounter++;
                    visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                    AddNextMove();
                    context.fillStyle = "white";
                    context.font = "bold 16px Arial";
                    context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                    context.fillText(MoveCounter, x - 6, y + 14);

                    DrawToSecond(x, y);

                    MovesMade[MovesMadeCount] = 2;
                    MovesMadeCount++;
                }
            }

            if (nextmove == 3) {
```

```
            if ((x == canvas.width * (19 / 20)) || (structure[((x + (canvas.width / 20)) / (canvas.width / 10))][((y + (canvas.height / 8)) /
(canvas.height / 4))][((x + (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))] == 1)) {
                //stack.pop();
            } else {
                context.save();
                context.globalCompositeOperation = 'destination-out';
                context.beginPath();
                context.arc(x, y, radius - 2, 0, 2 * Math.PI, false);
                context.fill();
                context.restore();
                context.fillStyle = "black";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);
                x = x + (canvas.width / 10);
                context.beginPath();
                context.arc(x, y, radius, 0, 2 * Math.PI, false);
                context.fill();
                MoveCounter++;
                visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))]++;
                AddNextMove();
                context.fillStyle = "white";
                context.font = "bold 16px Arial";
                context.fillText(visited[((x - (canvas.width / 20)) / (canvas.width / 10))][((y - (canvas.height / 8)) / (canvas.height / 4))], x - 12, y
- 4);
                context.fillText(MoveCounter, x - 6, y + 14);

                DrawToSecond(x, y);

                MovesMade[MovesMadeCount] = 3;
                MovesMadeCount++;
            }
        }

        MovementNo++;

    }
```

```
            //RedrawMaze();
            //alert("done");
        }
    </script>
</body>

</html>
```

Code Overview

Key variables

There are a number of key variables within the implementation section that will be explained here.

- x : Gives the x coordinate, in pixels, of the centre of the counter.
- y : Gives the y coordinate, in pixels, of the centre of the counter.
- Context : this variable represents the top maze, which has the counter drawn onto it.
- Context2 : this variable represents the bottom maze, which has the path taken by the counter drawn onto it.
- Structure[][][][] : Stores whether or not a wall exists between two coordinates, at the memory location of those coordinates. e.g structure[1][1][1][2] will return 1 if there is a wall between (1,1) and (1,2) and a 0 if there is not.
- Trees[][][] : This structure stores information about the locations of the walls within the maze. Trees are stored as a string of coordinates, with lines drawn from the first coordinate, to the second then to the third, etc. e.g Trees[0][0][0] gives us the x coordinate of the first point of the first tree, Trees[0][0][1] give the y coordinate of the first tree and trees[0][1][0] gives the x coordinate of the second point of the first tree. In summary, Trees[TreeNo][PointNo][X or Y coord].
- FLOATING : This variable defines whether the maze being generated will have floating walls or not.

List of Functions and their uses

- ReadCookie(name) : This function finds the cookie with the name inputted and returns its value.
- IsOnWallTree(X,Y) : Returns true if the inputted point is part of a tree that is connected to the wall.
- Valid(X,Y) : This function checks a wide range of criteria to decide whether or not the point about to be added to the current tree is allowed to be added to the current tree. This includes things like whether it is inside the canvas, whether it crosses another tree attached to the wall when it itself is part of a tree attached to the wall, etc.
- setCookie(c_name, value, exdays) : Sets a cookie with the name, value and duration inputted as parameters.
- eraseCookie(name) : This erases the cookie with the name inputted.
- FloatOnOff() : Switches the FLOATING variable to false if it is true, and true if it is false.
- SaveMaze() : Saves the maze under the name currently in the textbox.
- LoadMaze() : Loads the maze currently selected from the drop down box.
- RedrawMaze() : Resets the current maze.
- DrawToSecond(NX, NY) : Draws a line to the inputted point from the last point inputted onto canvas2.
- AddNextMove() : Adds the current move to the top or bottom of the cell the counter is currently in, adding to the list that is already there or starting one if there is not.

- doKeyDown(e) : This function runs when the user enters a keypress.
- LeftOpen(dir), FrontOpen(dir), MoveDir(dir) : Explained at the end of this section.
- FollowWall() : Runs the follow wall algorithm.
- RandomSolve() :Runs the random solve algorithm.

Testing

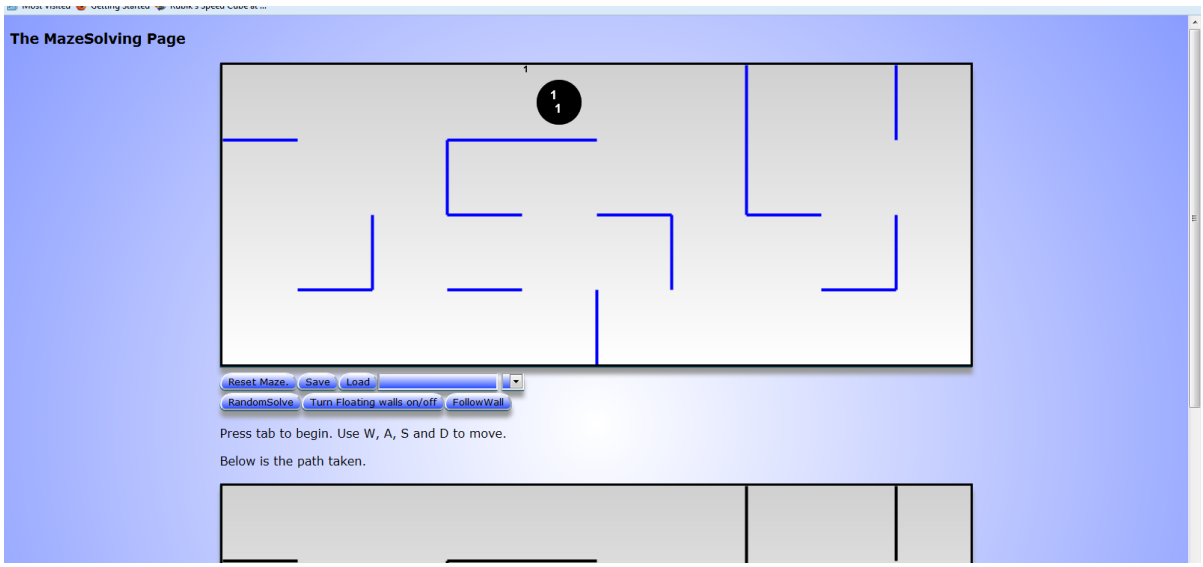
Before this tool can be used, it needs to be tested to confirm that it works as it was intended to. This will be done by checking all of the features it offers, to check they work as they should. This process is shown below in the table.

Test No.	Description and Expected Result	Observation	Notes
1	A random valid maze is produced every time the page is refreshed, with every cell accessible.	The page is refreshed 10 times and result is expected each time.	
2	The counter will not pass through a wall.	As expected.	
3	The counter will not pass through the outer wall.	As expected.	
4	When the counter moves to the next cell, the cell information is updated correctly	As expected.	When cell is visited more than 10 times, information overflows into the cell to the right or off the maze.
5	The path shown on the second maze is displayed at the same time as the upper maze is updated, and path is drawn correctly.	As expected, a whole solution is inputted, path is drawn as expected.	Path is only allowed to shift so far, after 21 shifts, no more shift is possible.
6	A random starting position is used each time a new maze is generated.	10 mazes are tested, a range of positions are used.	
7	When floating walls are turned on, floating walls are generated each time.	50 mazes are tested, with only 1 maze being generated without floating walls.	This appears to be a bug in the program, resulting in a rare generation without floating walls. Although this was investigated extensively, no solution could be found.
8	When floating walls are turned off, mazes are generated each time without any floating walls.	50 mazes are tested, with only 2 mazes being generated with floating walls.	Another bug leading to a rare false generation. This bug seems to produce errors slightly more frequently than the last.
9	Reset maze completely clears the maze and returns the counter to the same start position as it started on.	3 mazes are tested with 3 resets, all as expected.	
10	Mazes can be saved, and then	3 mazes are	

	appear in the dropdown list under the name they were stored with.	stored, as expect they appear in the dropdown menu.	
11	Mazes that have been saved can all be loaded separately.	The 3 maze are all loaded back as expected.	
12	A number of different solutions on the same maze are stored, and then reloaded.	Not as expected, only one solution per maze appears to be possible.	This will be important to point out in the user manual as a limitation of the solution.
13	The RandomSolve solves the maze completely as we would expect.	The maze is solved completely, given that the user inputs no moves before the procedure is run.	This algorithm solves every maze in exactly 79 moves, one less than double the number of cells. This is an interesting result.
14	The FollowWall button runs the algorithm as expected.	This works as expected.	
15	The page is run in Firefox, Internet Explorer and Google Chrome.	All features work in Firefox, most in Google Chrome and almost no features work in Internet Explorer versions 8-10.	Chrome was unable to support cookies. Internet Explorer 8 was used, this did not support the HTML5 features that the page uses. This includes the canvas tag. Internet Explorer 10 was also incompatible.

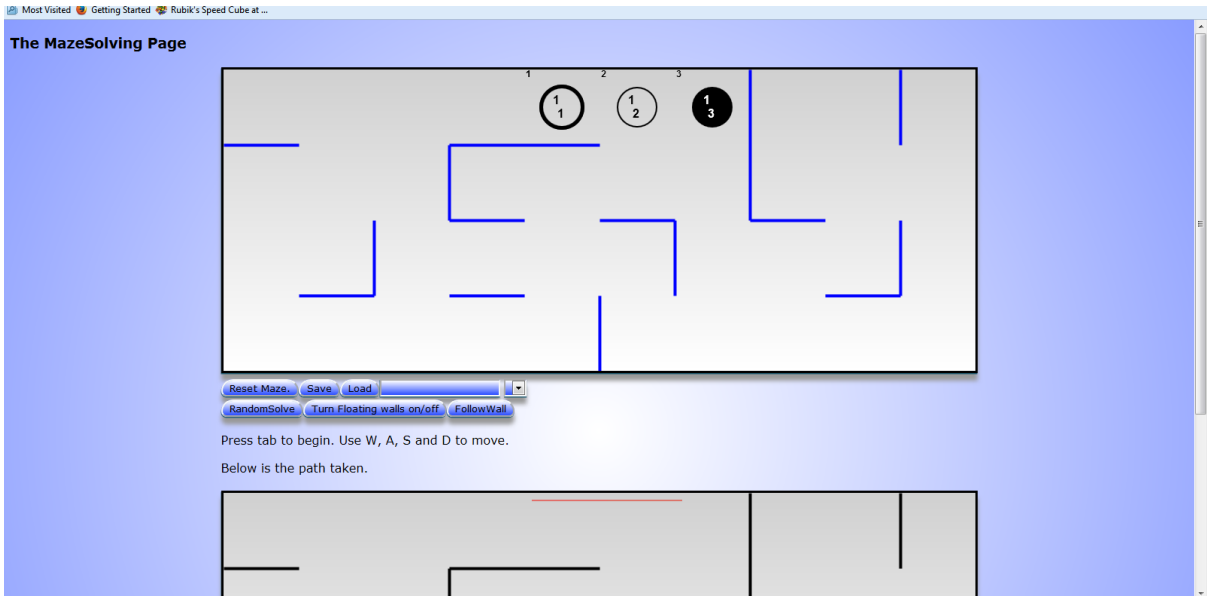
Screenshots

Test 1



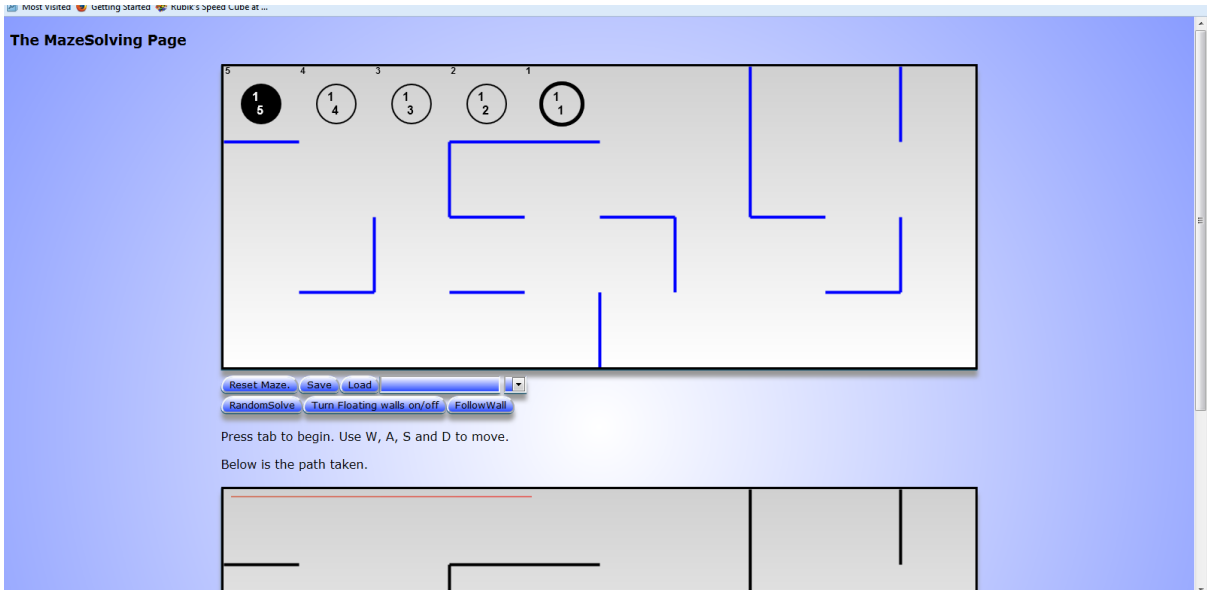
This screenshot shows a random valid maze being produced, with every cell accessable.

Test 2



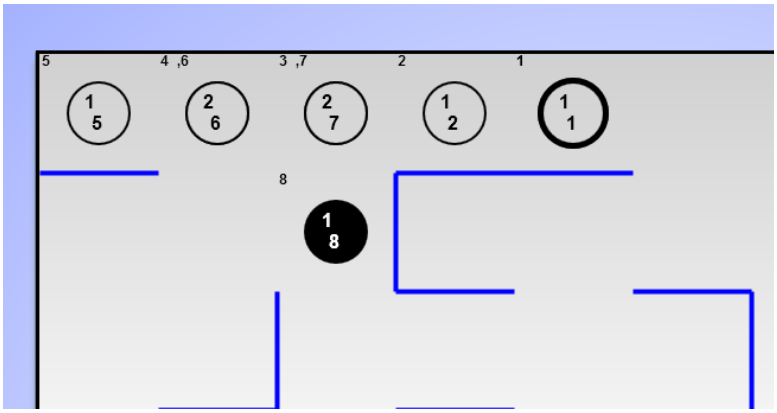
This screenshot shows a counter being unable to pass through a generated wall.

Test 3



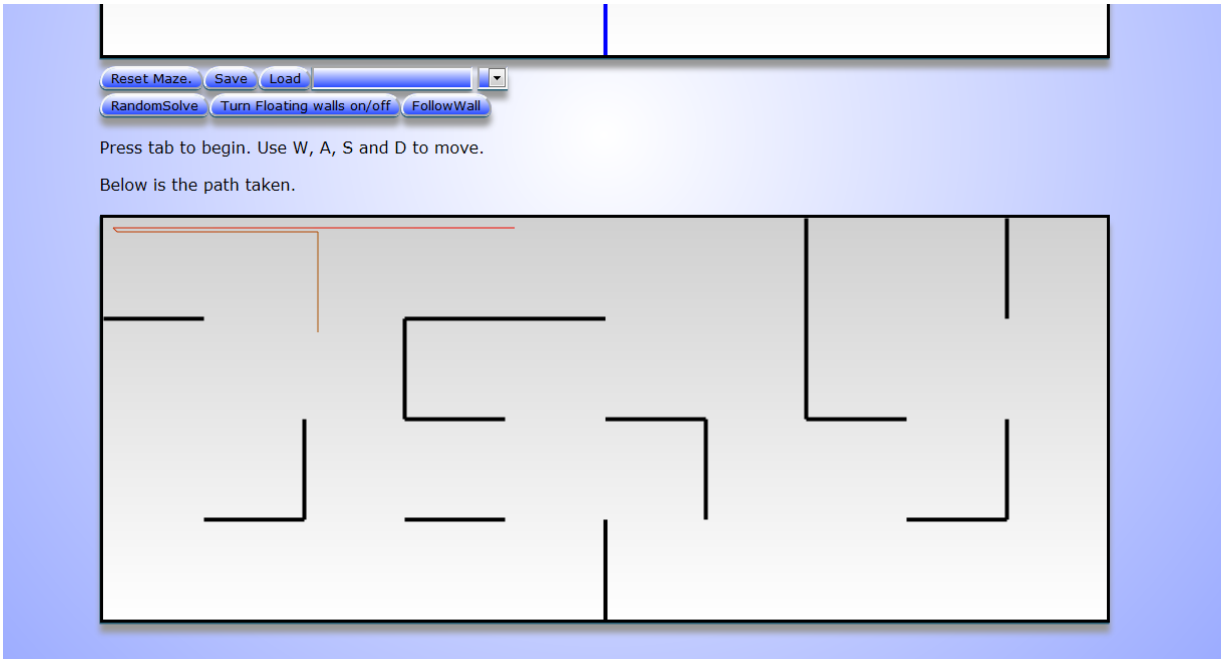
This screenshot shows a counter not being able to pass through the outer wall

Test 4



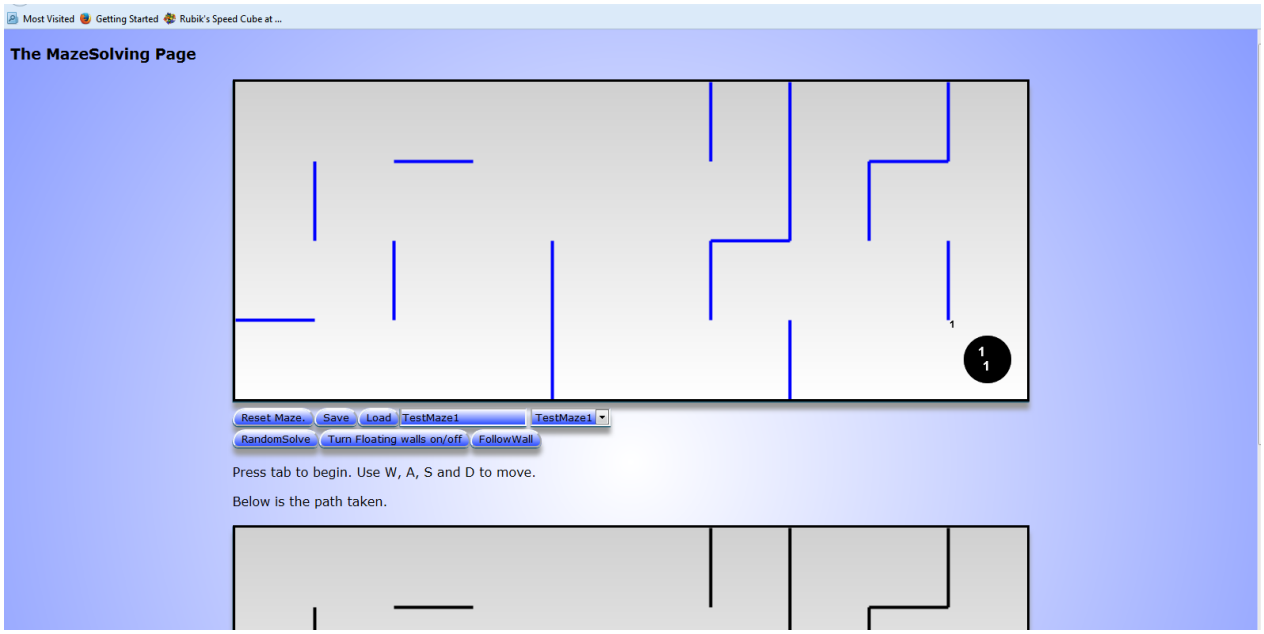
This screenshot shows information being correctly updated, moves being added to the top correctly and other cell information correct.

Test 5



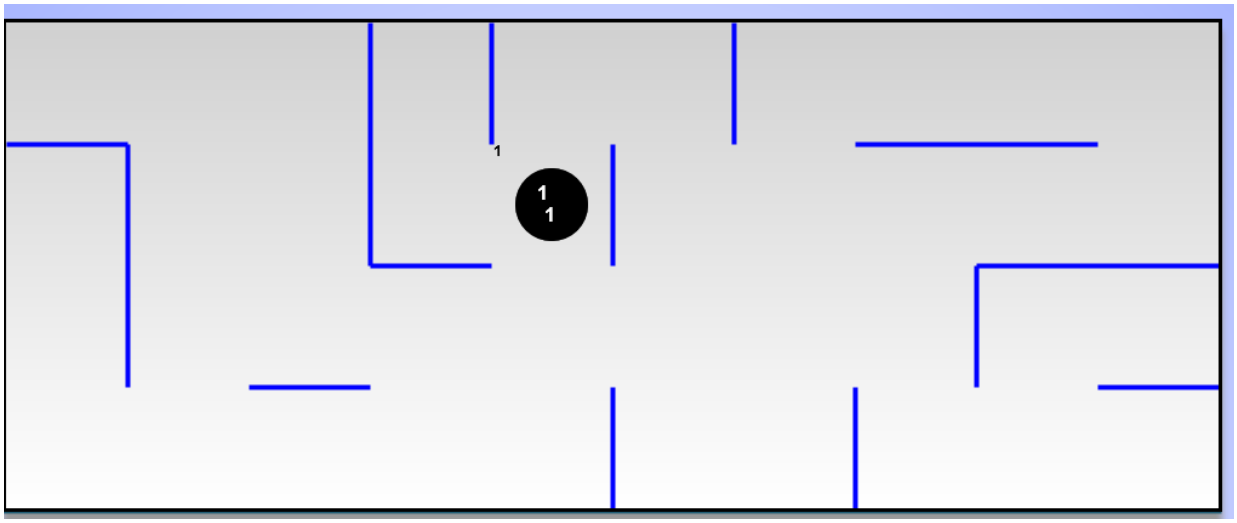
This screenshot shows the path being correctly drawn onto the second maze.

Test 6



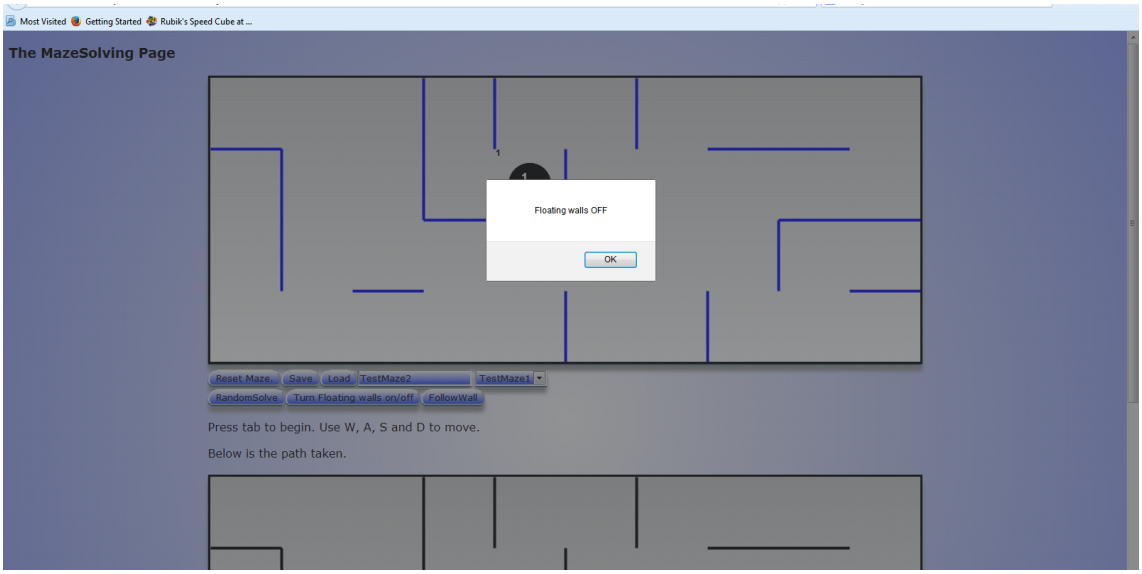
This screenshot shows a random starting position being produced each time, since this starting position is different to the one in test 1.

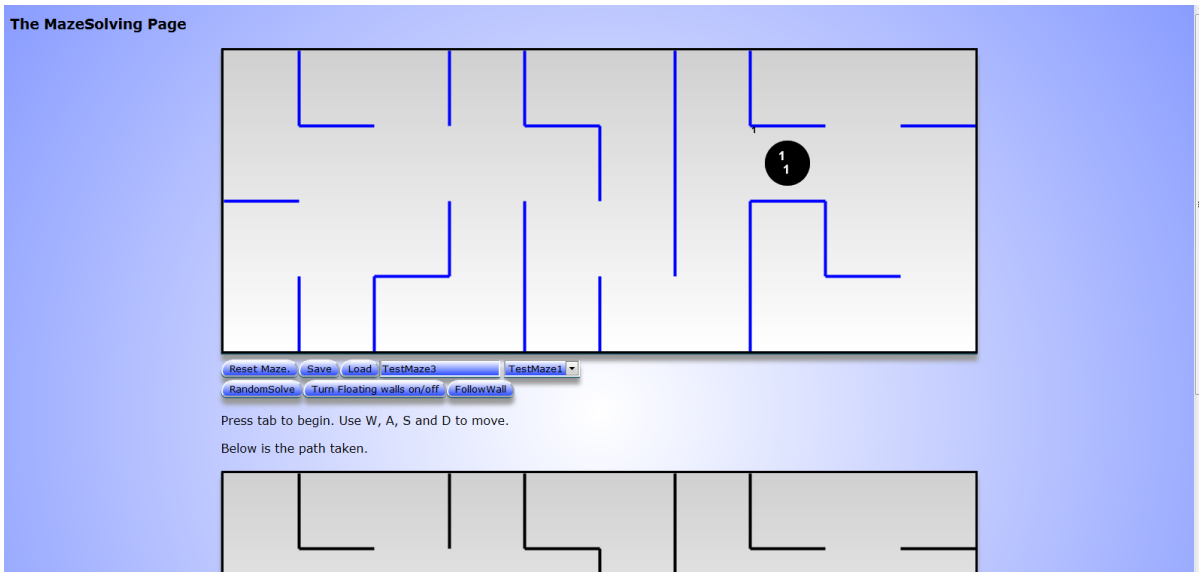
Test 7



This screenshot shows that floating walls have been generated.

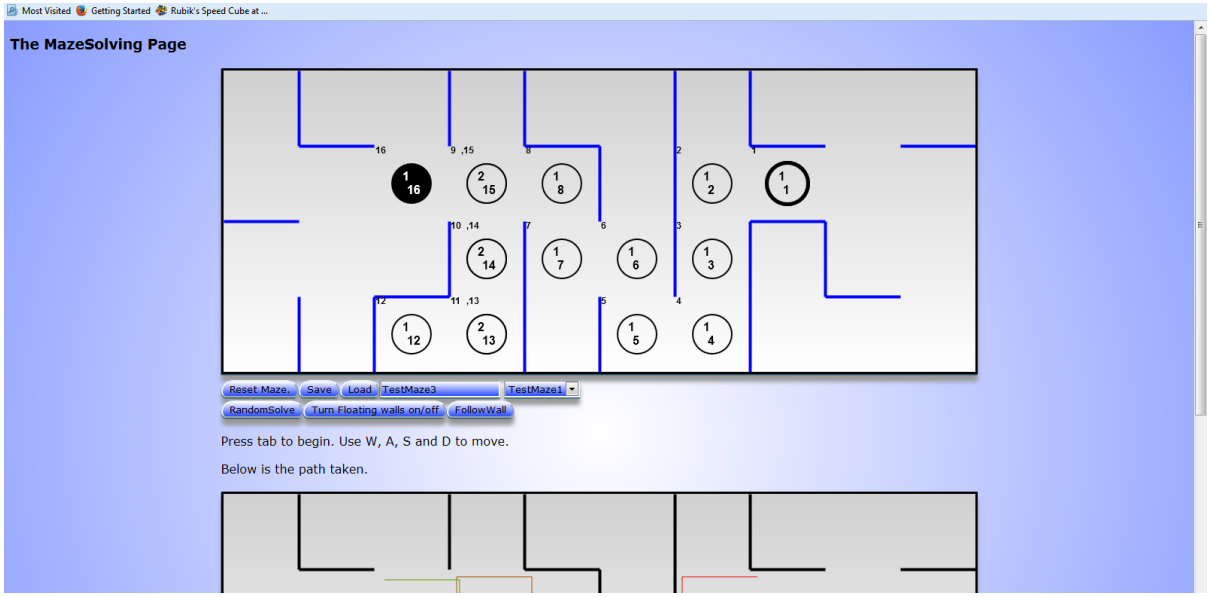
Test 8

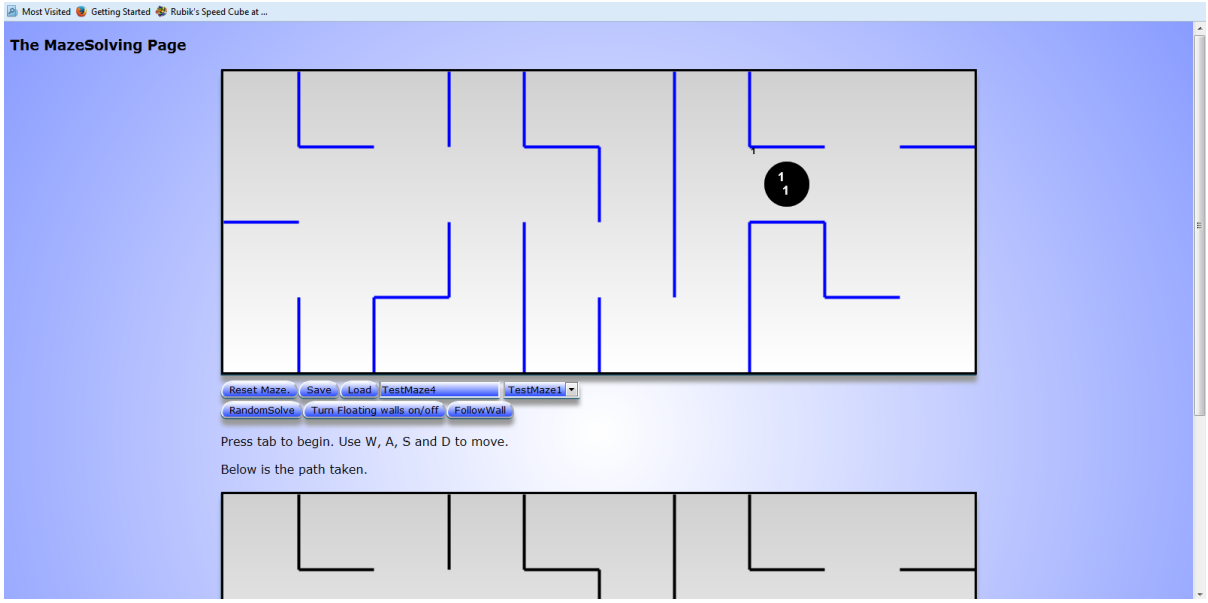




These screenshots show the maze being generated without floating walls.

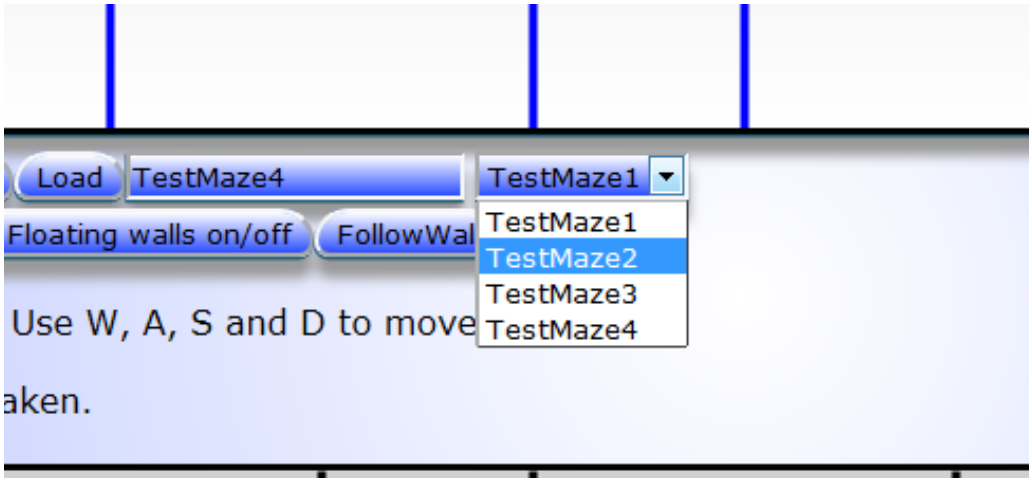
Test 9





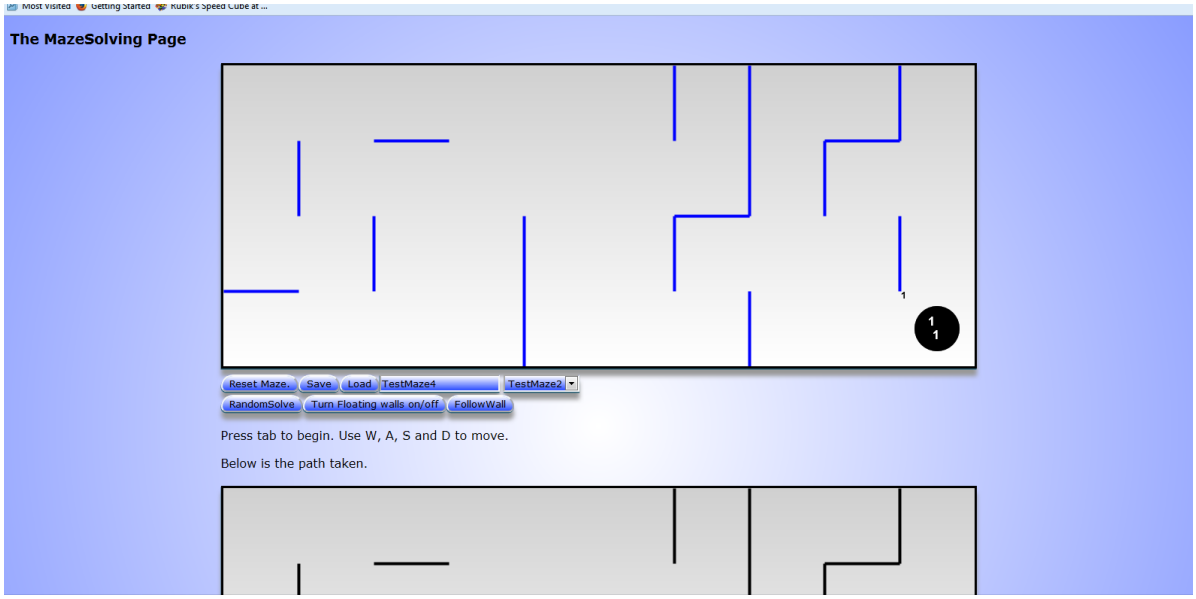
These screenshots show a counter first taking a path, and then that path being cleared and the maze reset.

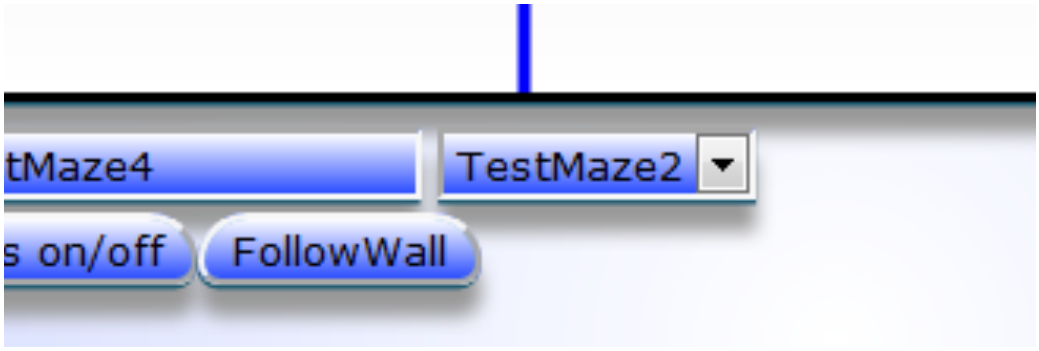
Test 10



This screenshot shows saved mazes saved under different, user inputted, names.

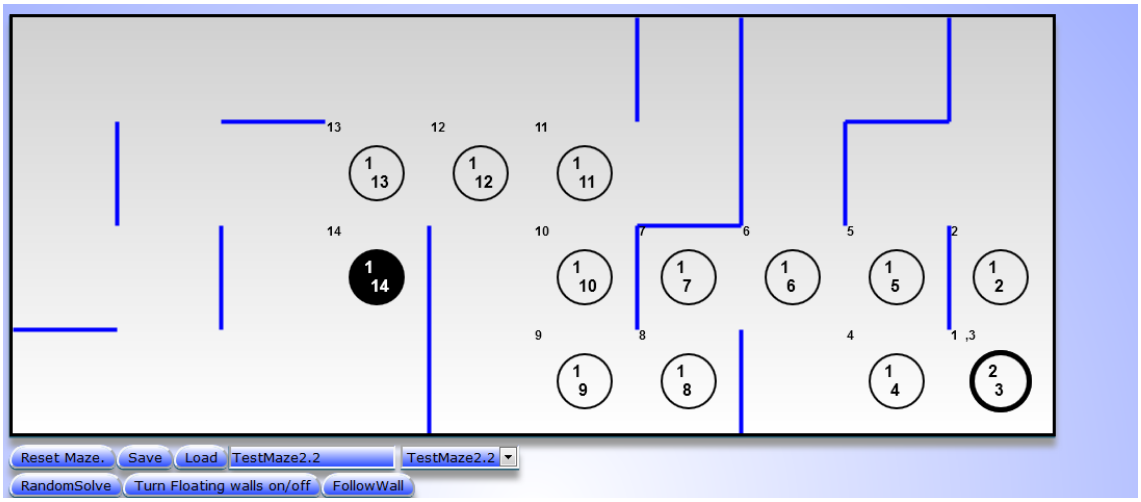
Test 11



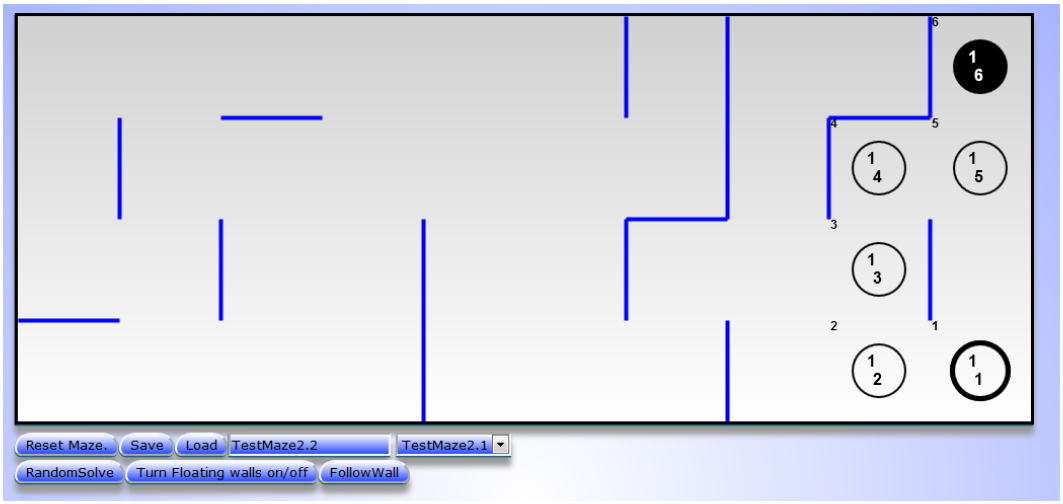


This screenshot shows test maze 2, the maze used in test6, being reloaded.

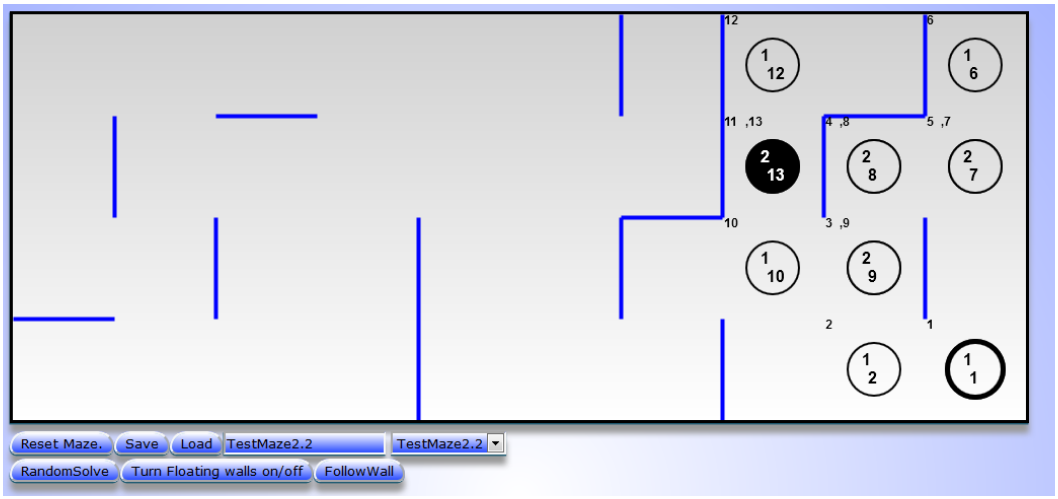
Test 12



This screenshot shows the path on test maze 2.

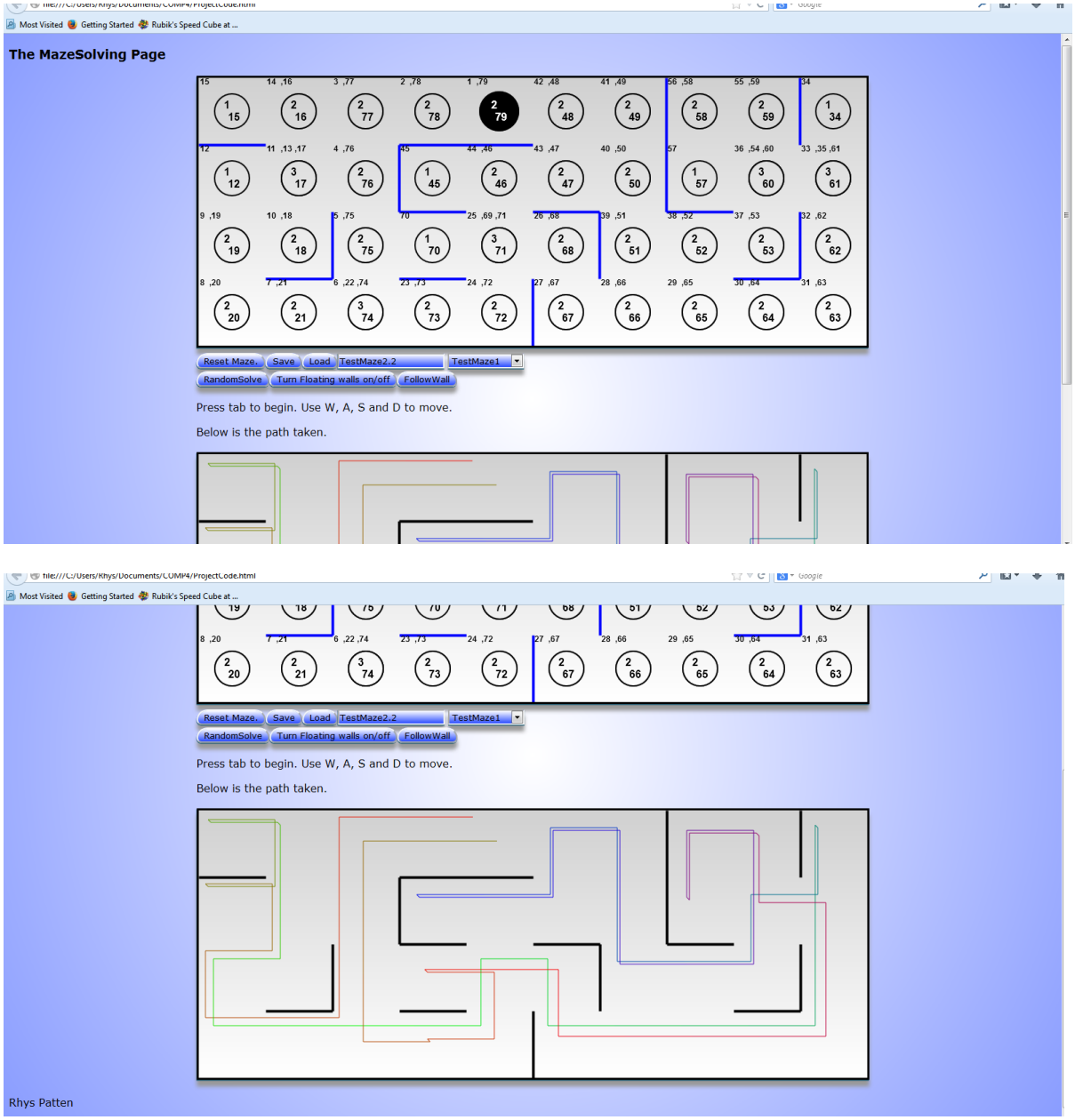


This screenshot shows a second path on test maze 2.



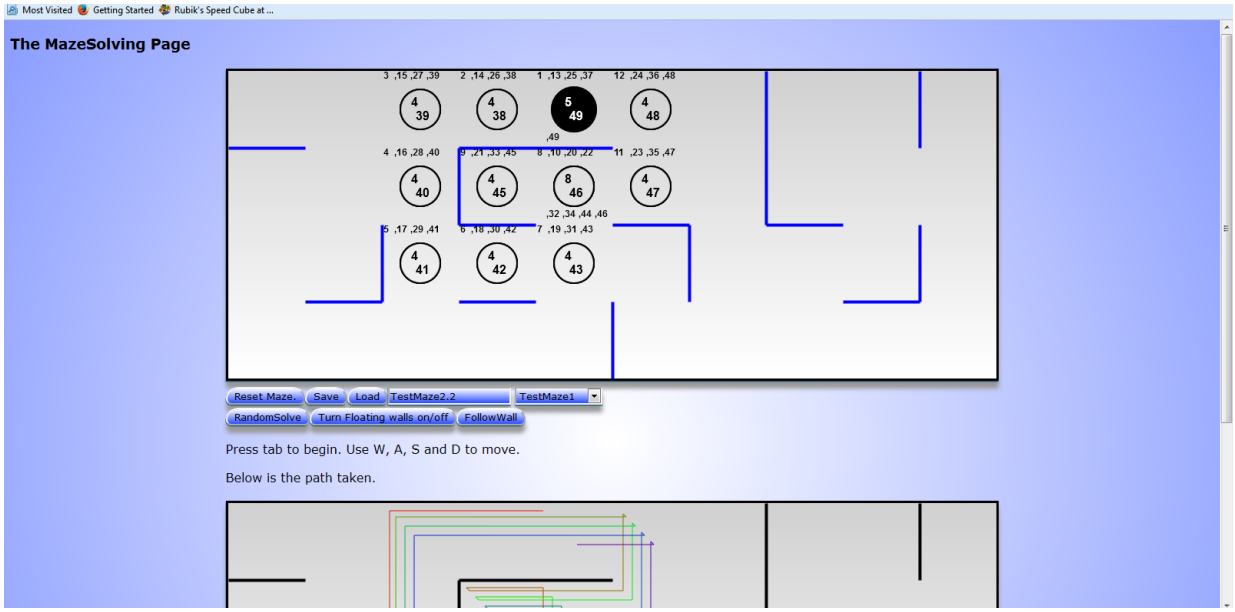
This screenshot shows the first path failing to load correctly.

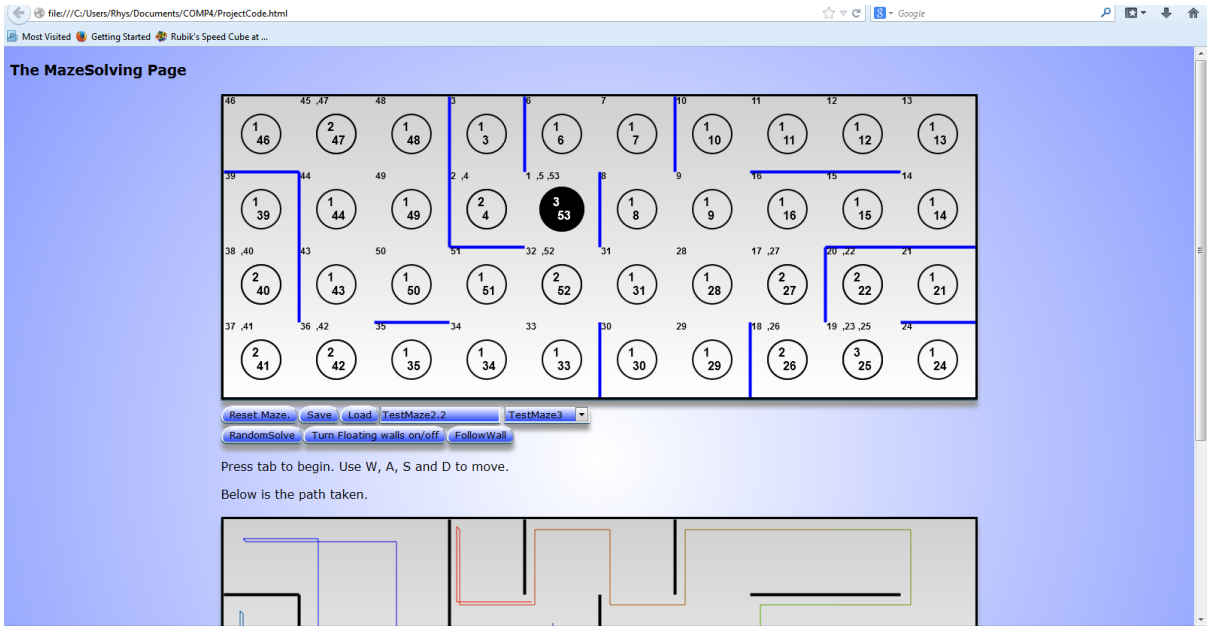
Test 13



These screenshots show the random solve algorithm working.

Test 14

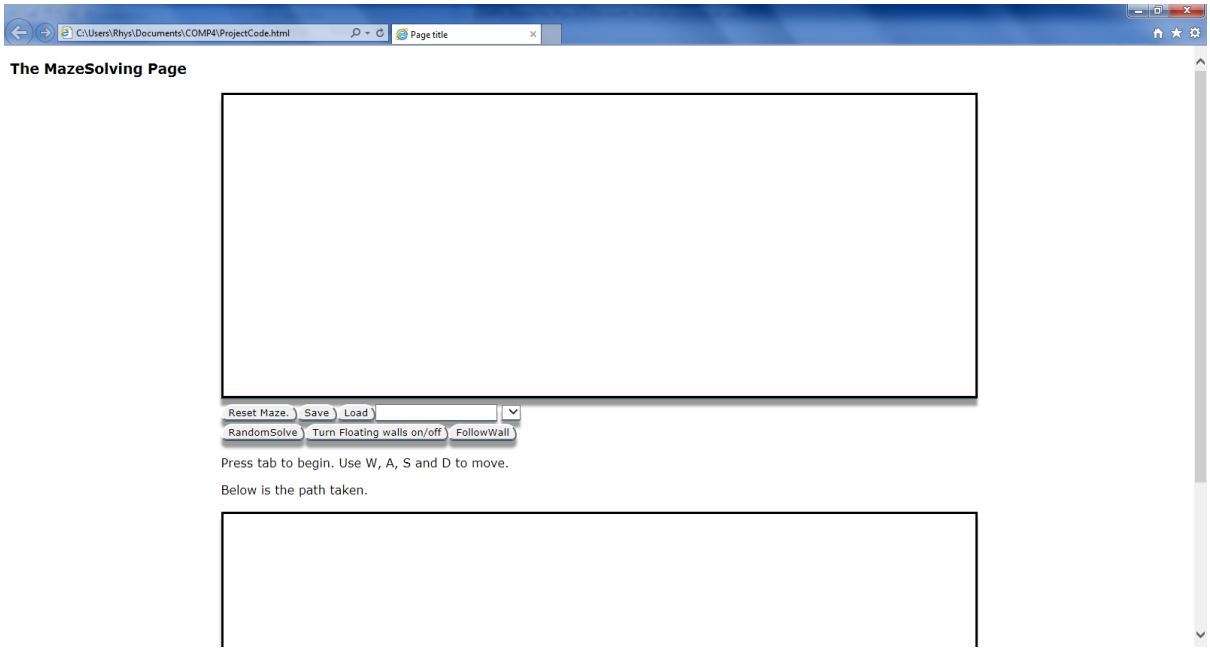




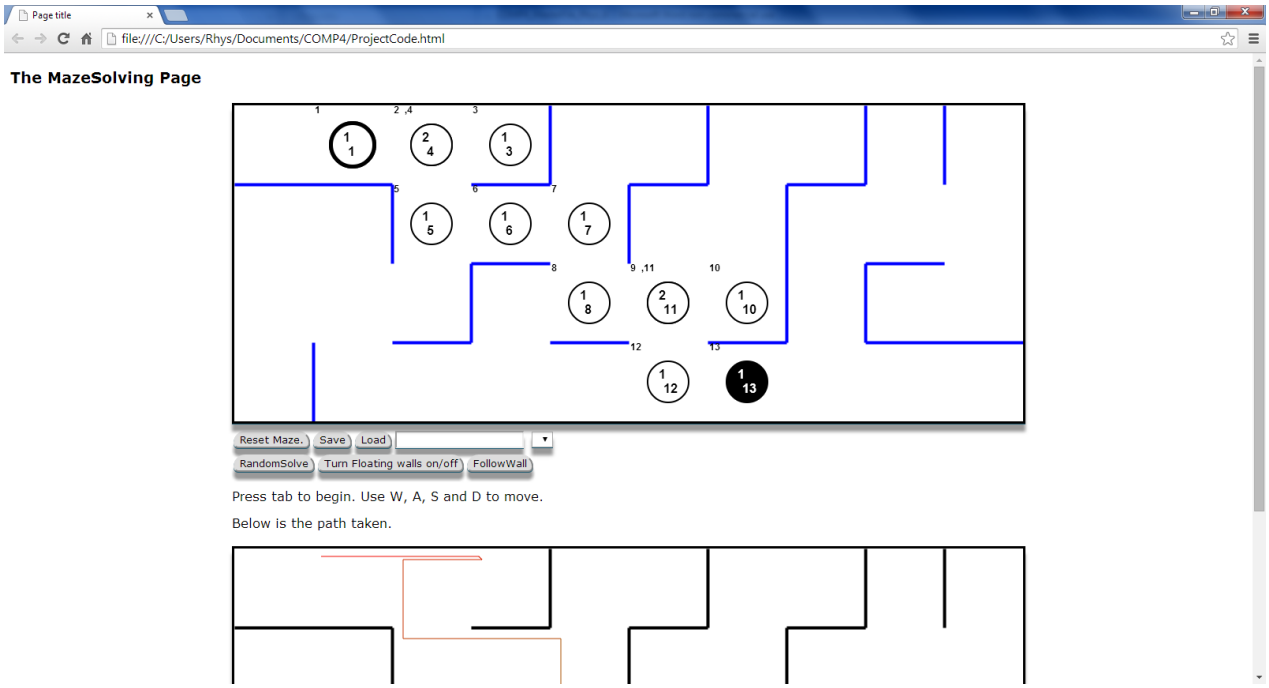
These screenshots show the follow wall algorithm working correctly.

Test 15

Internet Explorer 10:



Google Chrome:



These screenshots show incorrect loading in both Internet Explorer and Google Chrome.

Evaluation

Objectives

Below a table of objectives is show, with objectives highlighted in green, yellow or red to indicate full, partial or no achievement of that objective.

Objective	Evaluation of objective
The first objective of this system would be to produce a random valid maze. To ensure that the mazes produced are valid, all the points in the maze need to be accessible by the virtual robot. This would mean that no areas of the maze would be wasted, e.g. walls not stopping access to one half of the maze.	This objective was fully met, and the success of this feature of the tool is shown in the testing section. It shows that valid mazes are always generated, with no wastage.
The maze would also need to be generated with at least one floating wall to comply with the rules of the competition.	Although not as reliably achieved as the last objective, the failure rate here is very low. Failure appears to occur due to a low level bug in the algorithm used to generate the maze. Although this has been extensively investigated, no cause in the code for these errors could be found. A button to turn floating walls on and off was added to improve upon this objective, as was suggested in initial interviews with prospective users of the tool.
The second objective would be to allow users to test either own algorithms by manually exploring one of the random mazes. They will be able to move their robot around the maze on the screen, imputing each move and watching it move from cell to cell.	This objective was fully achieved, with a counter feature fully implemented and functional in the tool. The objective was achieved using key presses, but it might be possible to extend this and achieve this objective through touch, although this would increase hardware requirements, as a touch sensitive screen would be required for this feature to be available.
The number of moves being counted as it moves through the maze will be counted to give an idea at the end of how efficient the algorithm was, comparing it to the optimal solution which would need to be found.	A move counter was fully implemented, with additional information, such as number of times each cell has been visited, if it has been, displayed. The part of this objective to find the optimal solution to the maze was not completed. This was because it was not possible to find an algorithm to efficiently give the optimal solution, due the intractability of the problem, this is explained below. However, it turns out that the "follow wall" procedure does give the optimal solution if it completes the maze fully. The result of this is that if users do want an optimal solution, they would need to search for a maze that was full solved by the follow wall procedure.
The next objective is to allow, using an inbuilt algorithm, the user would be able to see the	Two inbuilt algorithms have been implemented, a random solve and a follow wall algorithm. The

efficiency of these algorithms to compare to their own. A number of algorithms could be simulated in this system (for example – random walk, …). The system would show the robot moving around the maze on its own(with delays between moves), giving reasons for each move to the user (e.g. North and south are open, just came from north, going south).	random solve guarantees a full random solve, but not a very efficient solve. The follow wall procedure shows exactly what the name of the procedure suggests. These algorithms are interesting as they have been commonly used by the end users in previous Robocup competitions.
A method of storing mazes and the current best number of moves they can be solved in and previous paths taken.	This objective was achieved, but as the testing showed, this feature does have some issues such as saving two different solutions to the same maze. However, these limitations have been highlighted in the user manual.
A method of allowing the user to input their own maze design, i.e. clicking where they would like their walls.	This objective was not achieved. This was because no way could be found in which to read in the maze from the user effectively. However possible solutions are discussed below.

Possible Solutions to allowing user inputted mazes

There are two key ways in which I see that it would be possible to achieve this, both relying on a way of the user inputting points to the page. Having run the function to input the maze, the first thing it would need to do would be to allow a user to click on a point on a grid shown over a blank maze. This would select the point. Either:

- The user just selects different walls to turn on and off, with clickable points being on walls. This would rely on users to validate their own inputted mazes.

Or:

- The user selects a point, the function then highlights valid points surrounding the current point, only allowing these points to be chosen or the tree to be finished by the user clicking on the current point of the tree again.

This second method would allow validation of mazes to be built into every user inputted maze, stopping any accidental invalid maze inputs from users. It is possible to detect the position of mouse clicks in HTML5 canvas tags, so after further research into methods, this would definitely be a potentially very large extension to the functionality of the tool. However, this further research and discussion has also highlighted to me the very high level of complexity of the function required to achieve this objective.

Optimal solution intractability

In order to prove that this problem is not solvable in polynomial time, and so is intractable, I will show that this problem is NP-Complete. I will do this by showing that this problem can be converted into the travelling salesman problem, a well know NP-complete problem. This is done by considering every cell in the maze to be a vertex, and there to be an edge between every pair of adjacent cells that has no wall between them. In this way the maze has been converted into a graph and solving

the travelling salesman problem will find the optimal solution to the maze. This is proof that finding the optimal solution is an NP-complete problem, and so is intractable. There are well known algorithms to get near optimal solutions in order to get around the TSM problem, but adding the feature stated in the objectives would only be worthwhile if the optimal solution could be found. A possible expansion would be to give a good solution using one of these algorithms and state whether or not it is optimal, since these algorithms can sometimes find an optimal solution.

User Feedback

Carl, a student using the tool in preparing for RoboCup 2014 gave this feedback:

“The application has a very clean and attractive design, with simple commands and layout. The numbering of the system allows for easy backtracking of the movements and the return to home function is very nice and clear, giving the user helpful information such as how many movements in total it takes to complete the maze as well as the visits per site. In addition to this, the ability to save past tracks is particularly useful and made simple.

There are, however, several areas which I, a user, feel should be addressed. Given the professional appearance, the comma error in the numbered listings is out of place and as of yet I have been unable to operate the custom movements - maybe a fault of the user or maybe instructions should be made more clear? Also, the user can repeatedly use "FollowWall", and multiple mazes superimpose; a bug, maybe?

Overall, I feel that the design is chic, well constructed and full of functionality. As soon as the aforementioned issues are addressed, this will be a very simple and handy application to use.”

End user Feedback

One of the objectives that was not met that we were looking forward to using but was not met was the ability to draw out our own mazes, however from conversation with Rhys, it was deemed Infeasible as these mazes could not be verified as competition ready.

Evaluation of Feedback

Carl’s feedback highlights the need to distribute the user manual, in order to make handing the software simpler. The design is approved of, but still needs refining slightly to get the tool to the point of being a finished product.

Possible System Improvements

The first improvement that could be made would be to make the user manual accessible online, either with a link to a pdf version, or a new button that would create a text box highlighting the key points of the user manual, perhaps also with instructions on how to find a full version of the user manual.

From my own observations of the project, I can see a number of things that I would do differently. The main change I would make in doing it a second time would be to use different data structures. By using dynamic data structures, such as a graph to represent the tree structure, the maze

generation could have been simplified. This would have led to a reduced error rate in generation. One of the main things that could be improved in the project would be the storing of information. At the moment, a huge amount of space is being wasted in the cookies being stored, and again this problem could be solved by using an abstract data structure. These changes would lead to improvements in the performance of the tool, leading to quicker generation and storage, as well as reduced error rates.

Appendix A

http://rcj.robocup.org/rcj2014/rescueB_2014.pdf