

**F454**

# **Computing Project**

Name: **Christopher Jones**

Candidate No: **9118**

Centre No: **65203**

# Table of contents

<b>Project Definition .....</b>	<b>3</b>
<b>Investigation and Analysis.....</b>	<b>5</b>
First interview.....	5
Existing System Solutions .....	8
Existing Maze Generation Solutions.....	15
Research into web technologies .....	19
Investigating hardware and software requirements.....	21
Second interview .....	23
Third interview.....	29
Draft Requirements specification.....	31
Final Requirements specification .....	35
<b>Nature of the solution .....</b>	<b>38</b>
Draft design objectives.....	38
Final design objectives .....	42
Design documents.....	45
Data structure design .....	51
Design of screen layouts .....	60
Contents of the help menus .....	69
Styling the user interface with CSS .....	70
<b>Algorithms .....</b>	<b>72</b>
<b>Test Strategy.....</b>	<b>91</b>
Alpha testing .....	91
Post development testing.....	92
End user testing .....	115
Acceptance testing .....	116
<b>Software Development.....</b>	<b>117</b>
Technical Description.....	117
Iterative development process .....	121
Complete program listings .....	208
<b>Testing.....</b>	<b>245</b>
Post development testing.....	246
Testing References .....	258
Response to post development testing .....	308
Repeats of unsuccessful tests .....	313
End user testing .....	318
Acceptance Testing.....	326
<b>User documentation .....</b>	<b>328</b>
<b>Evaluation.....</b>	<b>342</b>
Degree of success in meeting the original objectives .....	342
User evaluation .....	348
<b>Evaluation of user's response to the system.....</b>	<b>350</b>
<b>Desirable Extensions.....</b>	<b>351</b>
Good points of the system .....	351
Bad points of the system.....	351
Limitations of the system .....	351
Possible extensions .....	352

# Project Definition

## Client Definition

Jasmine Leftley, a sixth form student at Hazelwick School, has approached and asked me to develop a game that can be played frequently for small amounts of time (roughly five to ten minutes, although this can be flexible), provides an intellectual challenge (rather than to challenge their reactions) and has *replay value* (this means the game can be played regularly without getting boring).

Sixth Form students are often very busy with work, both at and after school hours. However, it is often suggested that these students break up their revision and study with small, five to ten minute breaks. As nearly all of these students have personal computers or mobile phones, which they actively use during the day, they often make use of them during their breaks to play *casual games*. This is why a student has approached me – in order to create a game that he and fellow sixth formers can play for small periods of in between breaks.

## End user definition

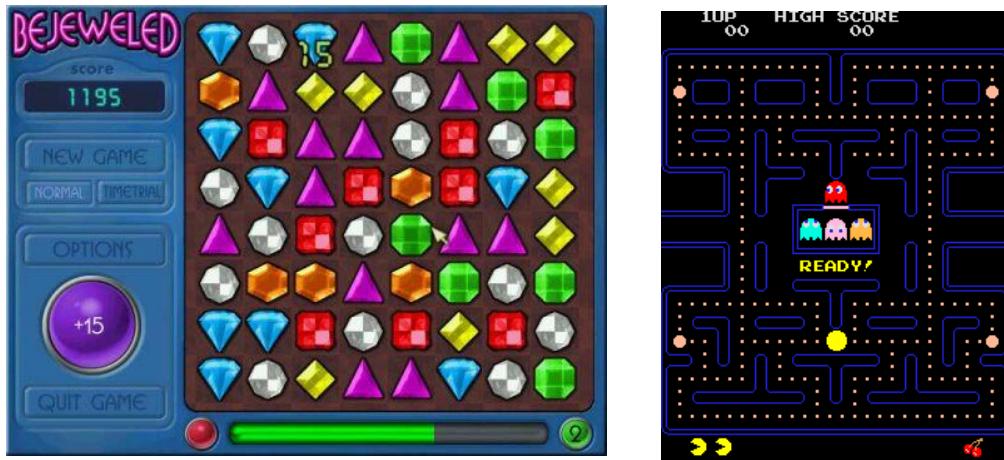
The end users of the game will be Sixth Form students who wish to play an intellectually challenging casual game that requires problem solving and logic. The end user group encompasses the client of the game, who will also play the finished solution.

## Current system

In recent years there has been a tendency towards people playing online ‘casual’ games, which are both easy to learn and play, fun and provide suitable challenge. However, many casual games have little replay value - once you have finished all the levels once or perhaps twice, they are no longer fun and interesting. Therefore, the game requested by the client students is in the genre of casual games with a high replay value.

The group of Sixth Formers, for whom I am designing a new casual game, already play casual games. However, they are bored by playing the same levels of games which once played have little replay value. They also requested a game that does not test their reaction times and instead tests their problem solving and logical abilities.

Examples of popular web-based casual games that already exist include **Bejeweled**, a game where the player swaps gems in a grid in order to form chains of gems that disappear until no more moves can be made. **Pacman** is a game where the player must avoid the moving ghosts for as long as possible and is considered the first casual game. Both have simple graphics and easy to learn gameplay.



Above: Images of the popular casual games Bejeweled (left) and Pac man (right).

### New system solution

The aim of this game is to provide an intelligent challenge that requires problem solving rather than simple repetition or reflex based interaction, and the game must have a high replay value.

### Sources of data

In order to gain further information in order to develop the new system I am going to conduct relevant data collection techniques and use this analysed data in order to gain a better understanding of the new system.

I will conduct several interviews with the client, who has lots of information and ideas regarding the new system she wishes me to develop. The first will be used to establish the main mechanic and style of the game he has in mind and the second will narrow the focus and produce information that can be used to model the specifications that come later in the project. Furthermore, the Internet will be used to gain understanding parts of the system that need to be established. This is referenced where appropriate.

Below is the signature of the client, which certifies that they have agreed to partake in providing guidance during the investigation and further stages during the development of the system. Note they will also need to produce their signature in order to confirm their agreements with different parts of the project – such as the requirement specification.

### Signature of client

# Investigation and Analysis

## Introduction

At this point in the project the client has given some vague information as to how she envisions the new system and further data collection techniques must be used to retrieve more information about the project. This information will then be thoroughly analysed in order to produce a requirements specification which will be used as an agreed upon reference as to the extent of the new system's functionality.

## Data collection methods

The data collection will be split into three stages. Initially, I will conduct an interview with the client in order to glean some more specific information on her ideas about the new system and the platform and end users who will use it. This will provide avenues for further investigation and analysis in the second stage, based upon discussions and conclusions made in the first interview and will use resources from the Internet.

A second interview will then be arranged in order to produce a narrower set of points that could be used to produce a requirements specification from which the rest of the development can be referenced. The second interview will use information gathered and analysed prior in order to form talking points around which discussion will be structured.

The advantages of interviews are that they can be used to collect a large amount of qualitative information and the nature of the interview can change direction over the course of a conversation. With a single client with whom I need to develop a greater understanding of the system, this is the best way in order to collect a large amount of information and better understand how this will the new system function.

## First interview

The first interview will be conducted with the client in order to discuss and gain a further understanding of the solution that the client wishes me to design and develop. I arranged via a verbal dialogue to meet the client after school in the study area in order to conduct this interview. At the moment, the constraints of the problem are vague, with limited details and mentions of a web based puzzle game for personal computers. This interview should help to narrow the focus and introduce some more concrete ideas on the problem. The transcript is shown typed up below.

### First interview transcript

*Me:* Thank you for attending this interview. I'm hoping that today we can have a discussion about the problem you approached me about earlier.

*Client:* That's fine with me.

*Me:* Could we start by you first describing to me why you approached me with this idea for a new system, which in this case is a casual puzzle game.

*Client:* Of course. In sixth form there is lots of time spent in free periods and [supervised] study sessions. I see lots of people with little to do when they need to relax in between finishing work. This is why I approached you [with the idea for the game].

*Me:* Do sixth formers play games currently?

*Client:* Some of them do. Most students are discouraged from playing games in their free periods and to do something constructive. However, I discussed with the head of sixth form a need for people to have breaks from their work. We discussed a game

I wanted to approach you about and it was agreed it would be okay to allow students to play a game that was a puzzler.

*Me:* A puzzler?

*Client:* Something which requires the player to use their brain, not their reflexes - and to be playable for small intervals of about five or ten minutes.

*Me:* This sounds like an achievable goal. So you want it to be run on the school computers?

*Client:* Yes the school computers would be necessary. I think it should also be usable on the computers that people bring in with them.

*Me:* Mobile devices? Phones? Tablets?

*Client:* Students are actively discouraged to use these to play games. The head of sixth form would not condone the use of these devices to play games. They're too hard to control people from playing in lessons for example.

*Me:* That's fine. So you think this should be available on a website?

*Client:* Yes. I envisioned with its own website. Lots of games I play have this format and I like it. It would make it easy for anyone to access. It wouldn't have to be installed on all the computers.

*Me:* Okay, thank you. If we can move onto the features or main mechanic of the game – have you had any ideas about that?

*Client:* Yes. I thought about a game where you have to solve a maze. So you have a top down view of a maze and you control something that has to move from a start point to an end point.

*Me:* That's an interesting idea.

*Client:* I'm worried that it would be difficult to create a lot of mazes for the game to remain interesting.

*Me:* I imagine it would be possible for the computer to create the mazes for you. I could investigate that for you. This would have the advantage of producing apparently random mazes.

*Client:* That sounds good. Sort of like how games like 'Minecraft' generate random worlds?

*Me:* Yes very much in that style. I'm sure that it's possible with mazes and I'll do some further research for you.

*Client:* Interesting. I thought about how every time the player completes one maze they then move onto a higher maze. This could keep the game fresh.

*Me:* Yes that's a good mechanic. Have you thought about how you would want the style of the game?

*Client:* Well when I thought about it I imagined a man running around a sort of stone walled maze like in real life. Then I thought about the style of 2048. Have you played it?

*Me:* Yes I have. I believe that has its own website.

*Client:* It does, it's really how I then envisioned this game to look and feel. Very blocky graphics and a subtle colour palette. Quite monotone if you know what I mean. Lots of greys and whites. You play a simple tile that moves around a maze of tiles with walls blocking the way and a tile you have to reach to end the level.

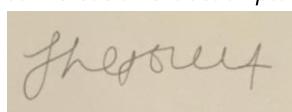
*Me:* That's very modern. They would be very appealing.

*Client:* Very simple graphics do appeal to me. I'd like this to come across in the finished system.

*Me:* I'm sure it will. I'm going to take the information I've gathered so far and do further research into mazes, the platforms available online and existing games that seem similar. I'm also going to do a second interview to get some more exact specification points. We can then agree upon a set of requirements.

### **Signature of client**

*The signature below certifies that the client agrees this is an accurate reproduction of the conversation that took place during the interview.*



### **Analysis of first interview**

This interview acted as a brief introduction into the problem area that the game needs to solve. Some useful information was gathered that has been beneficial in finding further areas to target additional investigation. Below are some points that summarise the important information gathered so far:

- A puzzle game needs to be created to satisfy the client's problem definition of a brainteaser game rather than a reflex-based action game.
- Needs to run on the computers at the sixth form.
- Needs to run on a range of PCs that sixth form students bring from home.
- Does NOT need to run on mobile or tablet platforms, as games on these are actively discouraged.
- The game needs to be embedded and accessible from a website.
- A game mechanic where the user has to solve mazes.
- Discussed a game mechanic where when the user solves one maze the next level will be a larger maze they have to solve.
- Possibly a system where the computer generates mazes for the user to solve as opposed to them being manually created by a developer.
- Modern and appealing graphics akin to that of casual games currently available.

It has become apparent I need to do further investigation in the following areas:

- The different platforms in which graphics can be rendered on a webpage will be investigated.
- Research into maze generation on computers and see if this is accessible in this context.
- Research into existing maze games.
- Find out the hardware and software specification of the computer's in the sixth form computer facilities and match that to the hardware and software specification
- A second interview with the client in order to gather more information in order to create a requirements specification.

## Existing System Solutions

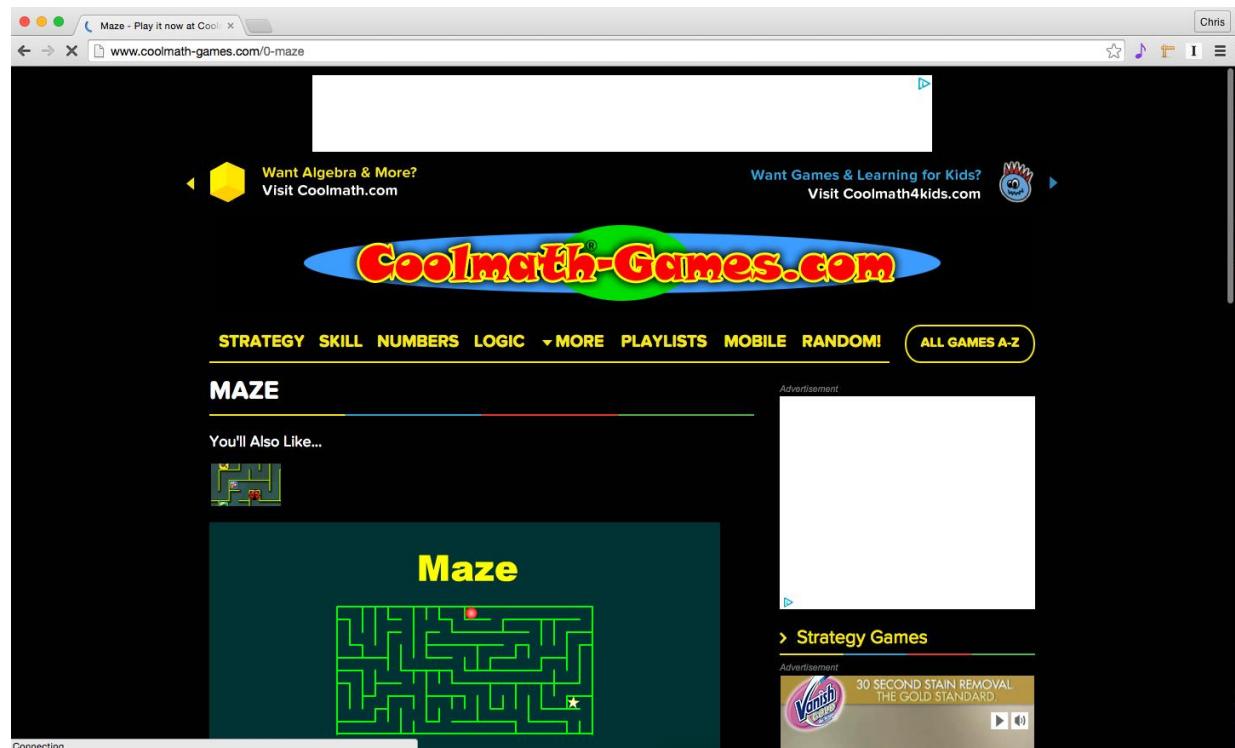
Following the first interview, it became apparent that some research needed to be completed on the existing solutions to the problem definition. Existing system solutions will be examined in order to determine the features that similar games have and the areas in which these games lack desirable features that could be included in the new system. This particular research will be used to structure the topics of the second interview in order to determine the features the client would like in the game by making sure it includes an ample number of features existing systems supply.

Games embedded on webpages that include mazes the player must solve will be used as existing systems. In order to locate these existing games I ran a web search with the keywords 'maze game'. After searching through the games that did not fit the style of game that had to be examined two suitable systems were found and are described below. Features that are desirable and undesirable are included as bulleted lists at the end of each section.

### 'Maze' – [www.coolmath-games.com](http://www.coolmath-games.com/0-maze)

<http://www.coolmath-games.com/0-maze>

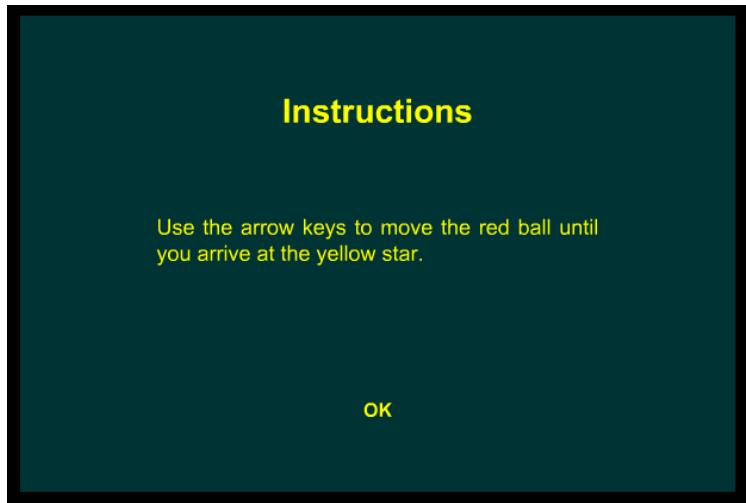
This is a simple maze game that is embedded on a webpage as an Adobe Flash application. This game will be used as an example of a current system solution that will influence decisions made about the new system. Below is an image which shows the website and the game's launch screen.



Below is another image of the full game's launch screen. It contains a start button that loads the game when pressed and an image of a maze like that of one in the game. This is shown in the image below.

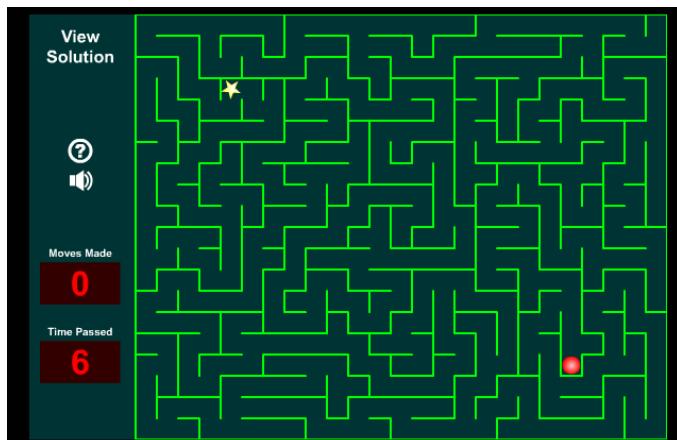


The following image shows the onscreen help in the form of short instructions that inform the user how to play the game. While these are brief instructions they are suffice enough to play the game with no worry for not understanding any features. It appears as if simplicity is at the core of the game.

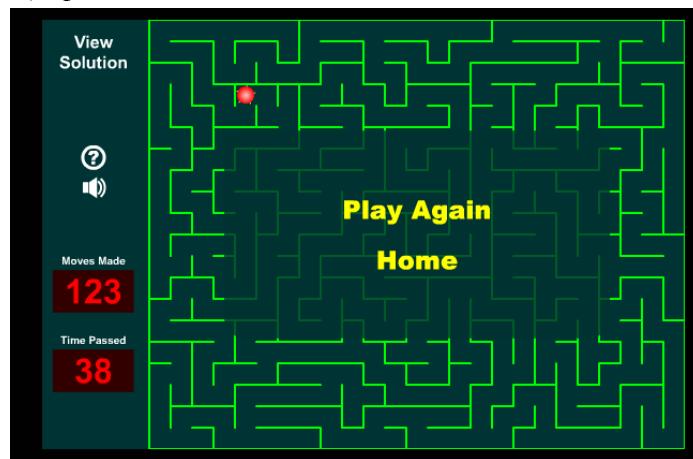


Below is an image of the player (the red ball) in a maze where the goal is a golden star. There is a display that shows the moves made and the time that has passed since the game has loaded on the left. There is also a link to the instructions should the user forget (the button labelled '?'). The game also contains audio and an option to mute it.

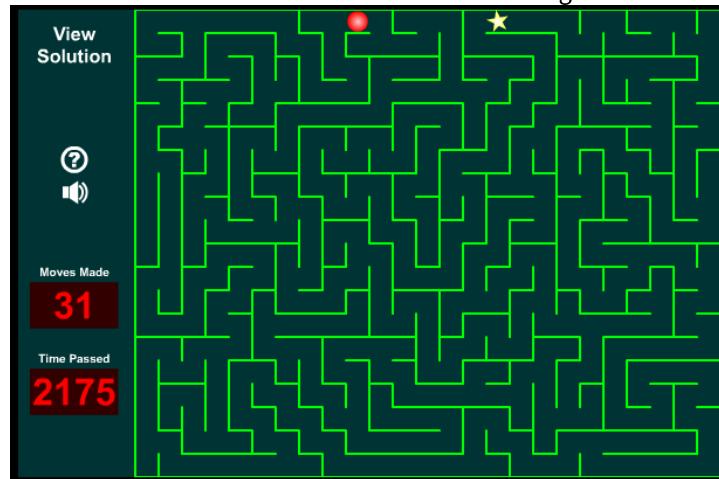
The arrow keys are used to move the ball and there is no real animation – the ball simply jumps



along rather than moving smoothly. Below is an image that shows what is displayed when the maze is solved and the ball is at the same position as the star. There is an option to return home to the launch screen or play again.



Playing again will generate a new maze the player can complete. Note the formation of the walls in the maze is different to the last one so it can be assumed that these mazes are generated. Note the mazes are the same dimensions. Another maze is shown in the image below.



Below is a summary of the desirable and undesirable / missing features of the game. This is so that the requirements of the new system can be correctly ascertained by identifying the features of the current systems that are desirable and avoiding undesirable features and including any potentially missing features.

### **Analysis of the first existing system**

#### Desirable features of the game:

- Hosted on a website so accessible via web browser on a personal computer.
- Mazes are randomly generated / change formation with each play.
- Displays a record of number of moves made.
- Displays the time taken to complete a maze.
- Contains onscreen help.
- Allows the game to be played repeatedly – there is no cap to the number of randomly generated levels that can be played.
- Contains the ability to have a solution to the maze presented to the user.

#### Undesirable / missing features of the game:

- Game is provided as an embedded Adobe Flash application. This is undesirable because it requires additional software to be installed to run the game.

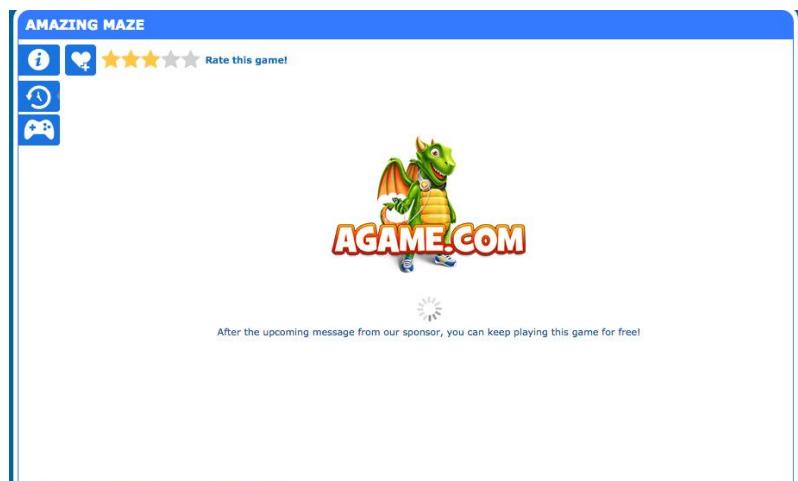
- Graphics appear very old fashioned and dated. Could be much more stylish to fit with the client's ideas from the first interview.
- Animation of movement of the player's circle is very jittery and makes game appear cheap.
- All mazes generated are the same size.
- There is no learning curve / the game stays at the same level of difficulty for the duration of play. Does not pose an increasing challenge.
- Only allows the use of arrow keys for input of movement. Provides no mechanic for use of WASD key layout popular for movement in games.

**'Amazing Maze' – [www.agame.com](http://www.agame.com/game/amazing-maze)**

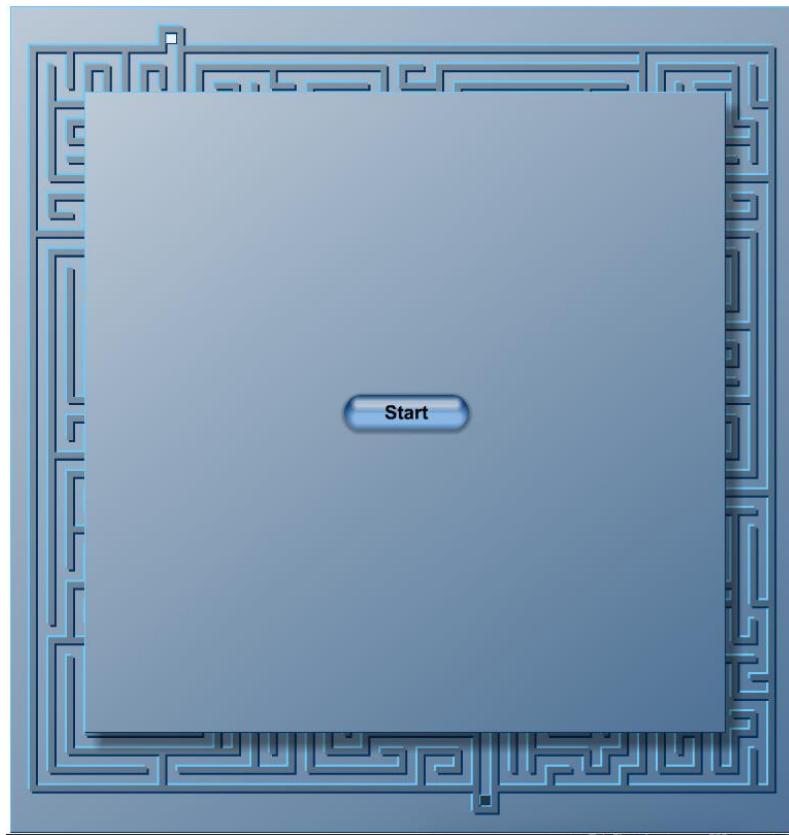
<http://www.agame.com/game/amazing-maze>

This is another online maze solving game that is embedded in a webpage, again as an Adobe Flash application.

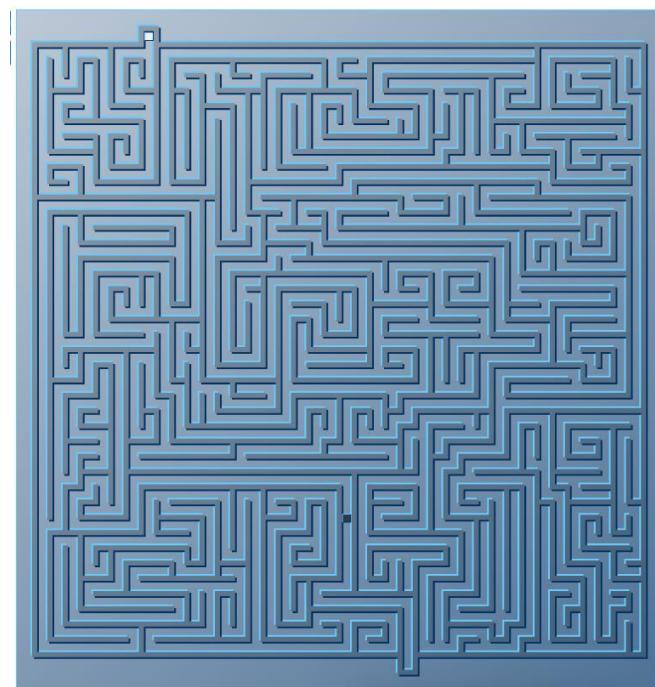
The game initially starts by displaying advertising before it is loaded. Although an image of any particular advert will not be included in this report, below is an image of a splash screen that displayed initially before the advert had loaded.



The game has no particular launch screen other than a large dialogue with a start button, as shown in the image below. No onscreen help is provided which makes it difficult to assess the necessary controls.



Upon pressing the start button, the maze is displayed. A white rectangle is displayed at the top and a dark grey rectangle at the bottom. Although it is not immediately obvious the white square is the player and the dark grey square is an opponent controlled by the computer. The dark grey square moves around the maze independently and attempts to reach the player's starting square and cause the player to lose if this happens. The player wins if they reach the computer opponent's starting square. This adds an additional and exciting level of complexity to the game. This is shown in the image below.



The graphics are preferable to those in the previous maze game although it could still be nicer. The controls are the same as the previous system, with the arrow keys allowing movement. The animation can at times be sluggish and the player seems to be able to glide over the corners of walls, which is not well designed.

### **Analysis of second existing system**

#### Desirable features of the game:

- Hosted on a website so accessible via web browser on a personal computer.
- Provides an exciting game mechanic where the player must reach the exit of the maze before the computer-controlled square reaches the player's starting square.
- Graphics are more stylish and an improvement upon the previous system examined.

#### Undesirable / missing features of the game:

- Game is provided as an embedded Adobe Flash application. This is undesirable because it requires additional software to be installed to run the game.
- Displays an advert before the game can be played. This is not a desirable feature of the client's.
- No onscreen help is provided. Makes understanding of controls and the game mechanic of an opponent very difficult initially.
- Design of menu is very simple and not appealing.
- Animation of player's movement is sluggish at times (although a lot better than the previous system examined).
- The game is very difficult, with a very steep learning curve (I was unable to beat the game).
- Only allows the use of arrow keys for input of movement. Provides no mechanic for use of WASD key layout popular for movement in games.
- The game becomes very 'laggy' with poor performance and response to input after a short amount of time.

### **Conclusion of existing systems**

With two existing system solutions having been examined, a list of features can be determined that may be discussed as possible features in the second interview.

#### Possible features that will be discussed in second interview

- Not use a proprietary technology or software that requires additional information / does not allow some platforms.
- Displays a record of number of moves made.

- Displays the time taken to complete a maze.
- The ability to automatically solve the maze.
- The inclusion of a computer-controlled opponent that also attempts to solve the maze.
- Simple but sufficient onscreen support as well as written documentation.
- Simple and aesthetically pleasing menus.
- Graphics and style so game appears more modern than the existing system solutions.
- Provide animations that are smooth and visually appealing.
- Allows the game to be played repeatedly – there is no cap to the number of randomly generated levels that can be played.

## Existing Maze Generation Solutions

In the first interview it was decided the premise of the game would involve the user solving mazes. It was discussed with the client as to whether these mazes could be generated by a computer or would have to be manually designed. This stage of the Investigation and analysis will look into different ways that computers can be used to generate mazes.

After some initial searching on the Internet it became apparent maze generation is a topic that has been thoroughly researched in the field of Computer Science. Therefore, there are already numerous algorithms that generate ‘perfect mazes’ (*a maze which has only one path from any point in the maze to any other point*).

Rather than reinvent the wheel and develop a completely new maze generation algorithm, an existing algorithm will be implemented in the new system. What follows are various popular maze algorithms and their implementations.

### (Randomised) Depth First Search Algorithm

*(All information on this maze generation algorithm was sourced from the information at the web address [http://en.wikipedia.org/wiki/Maze\\_generation\\_algorithm](http://en.wikipedia.org/wiki/Maze_generation_algorithm). This article was correct as 15/10/2014).*

This is a simple maze generation algorithm referred to as randomised depth first search or equally just depth first search. This algorithm is said to produce aesthetically pleasing mazes. Below is an image of a simple algorithm that describes how randomised depth first search is used with a recursive backtracking method.

#### **Recursive backtracker** [edit]

The depth-first search algorithm of maze generation is frequently implemented using **backtracking**:

1. Make the initial cell the current cell and mark it as visited
2. While there are unvisited cells
  1. If the current cell has any neighbours which have not been visited
    1. Choose randomly one of the unvisited neighbours
    2. Push the current cell to the stack
    3. Remove the wall between the current cell and the chosen cell
    4. Make the chosen cell the current cell and mark it as visited
  2. Else if stack is not empty
    1. Pop a cell from the stack
    2. Make it the current cell
  3. Else
    1. Pick a random unvisited cell, make it the current cell and mark it as visited

#### Advantages of randomised depth first search

- Easy to implement.
- Uses simple data structures, including a stack.
- Can be implemented using backtracking that is a simple algorithm that is well documented on the internet.

#### Disadvantages of randomised depth first search

- Mazes are relatively easy to solve.
- The algorithm can cause stack overflow issues on some computer architectures if a recursive method is used (the algorithm can be alternatively be written iteratively).

#### **Wilson's algorithm**

(All information on this maze generation algorithm was sourced from the information at the web address <http://bl.ocks.org/mbostock/11357811>. This article was correct as 16/10/2014).

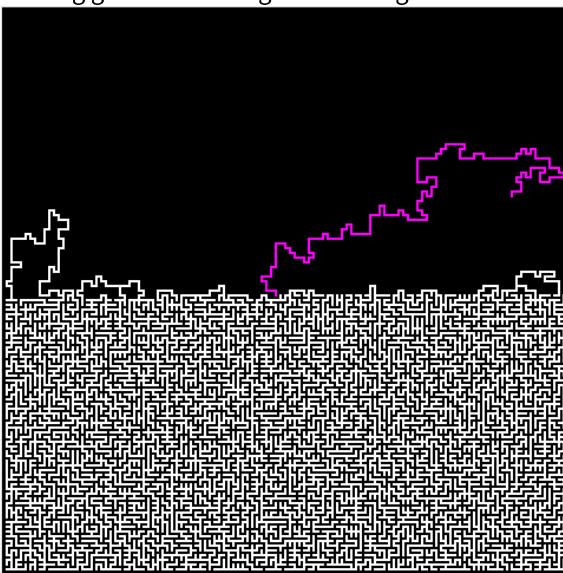
Wilson's algorithm uses [loop-erased random walks](#) to generate a uniform [spanning tree](#) — an unbiased sample of all possible spanning trees. Most other maze generation algorithms, such as [Prim's](#), [random traversal](#) and [randomized depth-first traversal](#), do not have this beautiful property.

The algorithm initializes the maze with an arbitrary starting cell. Then, a new cell is added to the maze, initiating a random walk (shown in magenta). The random walk continues until it reconnects with the existing maze (shown in white). However, if the random walk intersects itself, the resulting loop is erased before the random walk continues.

Initially, the algorithm can be frustratingly slow to watch, as the early random walks are unlikely to reconnect with the small existing maze. However, as the maze grows, the random walks become more likely to collide with the maze and the algorithm accelerates dramatically.

The global structure of the maze can be more easily seen by [flooding it with color](#).

Another famous maze generating algorithm is Wilson's algorithm. Below is a description of Wilson's algorithm from the website described in the source. Below is a screenshot of a maze in the process of being generated using Wilson's algorithm.



The source also contains a code sample of generating a maze using Wilson's algorithm. This is a HTML document with embedded JavaScript.

#### Advantages of Wilson's algorithm

- Mazes are considered ‘beautiful’ or aesthetically pleasing.

#### Disadvantages of Wilson's algorithm

- Generating a maze can be very slow.
- A more difficult algorithm to produce.

### **Recursive Division Algorithm**

(All information on this maze generation algorithm was sourced from the information at the web address [http://en.wikipedia.org/wiki/Maze\\_generation\\_algorithm](http://en.wikipedia.org/wiki/Maze_generation_algorithm). This article was correct as 15/10/2014).

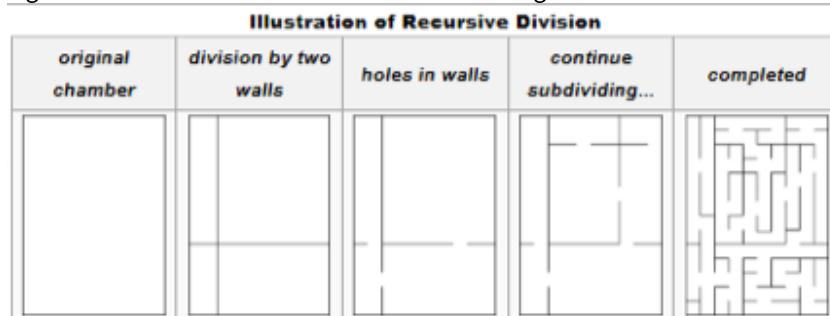
Most maze generation algorithms begin with every wall of every cell in the maze in place and the algorithm progressively carves a path through these walls in order to establish a maze. However, recursive division is different because it begins with no walls in place and progressively constructs them.

Below is an image of the description of the algorithm from the source website.

Mazes can be created with *recursive division*, an algorithm which works as follows: Begin with the maze's space with no walls. Call this a chamber. Divide the chamber with a randomly positioned wall (or multiple walls) where each wall contains a randomly positioned passage opening within it. Then recursively repeat the process on the subchambers until all chambers are minimum sized. This method results in mazes with long straight walls crossing their space, making it easier to see which areas to avoid.

For example, in a rectangular maze, build at random points two walls that are perpendicular to each other. These two walls divide the large chamber into four smaller chambers separated by four walls. Choose three of the four walls at random, and open a one cell-wide hole at a random point in each of the three. Continue in this manner recursively, until every chamber has a width of one cell in either of the two directions.

Below is a diagram that shows how the recursive division algorithm functions.



#### Advantages of recursive division

- Easy to implement a solution in this method.

#### Disadvantages of recursive division

- Mazes are easy to solve because it is obvious to see which divisions in the maze are easy to avoid.
- Mazes are not as visually appealing.
- Algorithm works best when used recursively (although an equivalent iterative method could be used). This could cause potential Stack Overflow errors.

#### **Decision on maze generation algorithm**

Should the client decide in the second interview that they wish for the mazes to be randomly generated, I intend to use the **randomised depth first search algorithm** in order to implement this feature. I am not involving the client in this decision as there area of expertise is in the definition of the problem area and not the implementation of the solution. However, I will still notify the client I have chosen this method for the purpose of transparency. I have chosen this solution because it is extremely well documented and should be simple to implement in the system. The mazes should be of sufficient difficulty and an iterative approach will be used to prevent any Stack Overflow errors (these are errors that can be caused in a recursive program in which a function calls itself more times than there is space for in memory).

**Further research into randomised depth first search algorithm**

Now that I have decided upon a method of randomly generating the maze it is necessary to collect further information on the topic.

I found a tutorial at <http://dstromberg.com/2013/07/tutorial-random-maze-generation-algorithm-in-javascript/> which contains a step by step guide in JavaScript that explains how to create a maze with a random depth first search algorithm. This link is correct as of 20/10/2014.

This tutorial is useful as the step by step procedure of writing code in JavaScript that produces the functionality required will be very useful both in the Algorithms section, in order to better help design and plan an algorithm, and in the Software Development section, where it can be used as a basis for the development of my own randomised depth first search code.

Below is an image of an excerpt from the webpage. More information and code samples can be obtained by following the link.

There are a number of different common algorithms for generating a random maze, and each approach will yield differing characteristics for the completed maze. For this particular project, I decided upon a depth-first search algorithm, with recursive backtracking. This is probably the simplest approach to the problem, and ensures that every maze will be solvable, and that every cell of the maze will be accessible, ensuring no wasted space in the maze.

The first step in constructing the algorithm is determining what is needed for the return values. For my game, I decided that I wanted a nested array that defined each cell, consisting of arrays of cells (x values) nested in arrays of rows (y values). Each cell would then be defined by the presence of each of its four borders. I decided to construct the result in this manner to facilitate an easy process of laying out each cell in HTML, and then defining each of its borders using CSS. As such, the border definitions are presented in CSS order of [ top, right, bottom, left]. Familiarity with HTML should make it apparent that defining rows as the first element of the array, with cells nested in rows, provides an easy way to loop through the array when drawing the final outcome. This may make it slightly confusing in that y values are stored before x values, but the alternative requires much more convoluted for loops than are really necessary. So the output of the function will be in the form of:

## Research into web technologies

The game will be accessed by personal computers via the Internet so a web technology will need to be used to program the game. The display of graphics will be necessary.

### Adobe Flash

Adobe flash is a browser plugin that targets a variety of web browsers and allows content to be displayed and programmed in ActionScript – a proprietary language developed by Adobe Systems. Adobe Flash was used by both of the existing systems in order to deliver their games to users – however this is not indicative evidence that it should be used in the new system solution.

#### Advantages of Flash

- Content display is independent of web browser – which means all content is displayed identically in all web browsers (providing they have a correct Flash plugin installed).
- ActionScript can be used to control graphics and accept input in order to develop the programming required to develop a game.

#### Disadvantages of Flash

- Requires an expensive suite of tools in order to develop for the platform (Adobe flash professional).
- Requires users to potentially have to download an additional browser plugin before they'll be able to access the new system solution.
- Can often take a long time to load that may cause users to leave the site.
- Support for flash has declined in recent years in favour of open source web technology.

### HTML5 canvas element

This is relatively new open source technology that can be used to dynamically generate content on the web. It is part of the HTML 5 specification that is conformed to by an increasing number of browsers. JavaScript, a programming language that executes in the client's browser, can be used to programmatically alter the contents of the canvas in order to display images and graphics. Below is an image taken from the WHATWG (Web Hypertext Application Technology Working Group) website which shows a description of the canvas element. WHATWG is a group that authors specifications regarding the HTML standard of content on the web.

This is sourced from the link <https://html.spec.whatwg.org/multipage/scripting.html#the-canvas-element> and is correct as of 28/10/2014.

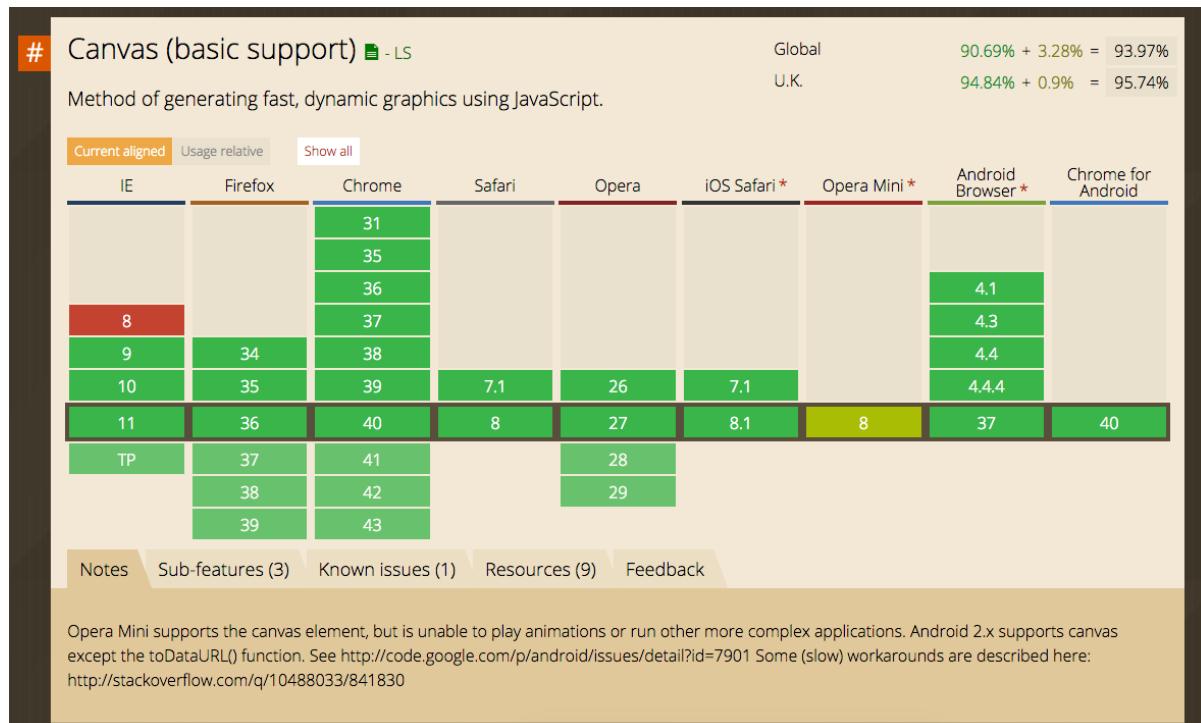
The **canvas** element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly.

Authors should not use the **canvas** element in a document when a more suitable element is available. For example, it is inappropriate to use a **canvas** element to render a page heading: if the desired presentation of the heading is graphically intense, it should be marked up using appropriate elements (typically **h1**) and then styled using CSS and supporting technologies such as Web Components.

When authors use the **canvas** element, they must also provide content that, when presented to the user, conveys essentially the same function or purpose as the **canvas**' bitmap. This content may be placed as content of the **canvas** element. The contents of the **canvas** element, if any, are the element's **fallback content**.

In interactive visual media, if **scripting is enabled** for the **canvas** element, and if support for **canvas** elements has been enabled, the **canvas** element **represents embedded content** consisting of a dynamically created image, the element's bitmap.

Below is an image from <http://caniuse.com/#search=canvas> (correct as of 28/10/2014), which shows the browser support for the canvas element. Note that browser support in most modern browsers is very strong, especially with versions of Chrome and Firefox. Correspondence with the client will need to take place in the second interview that determines whether support of the canvas is viable on the target software.



### Advantages of Canvas

- Specified by The WHATWG and as such is a certified standard supported by a reputable working group.
- Can be integrated directly into a webpage authored in HTML.
- Graphics can be controlled with JavaScript, which runs in the client's browser and is included with virtually all modern web browsers.
- Tutorial on randomised depth first search documented previously used JavaScript. Shows that maze generation is very possible with JavaScript and canvas.

### Disadvantages of Canvas

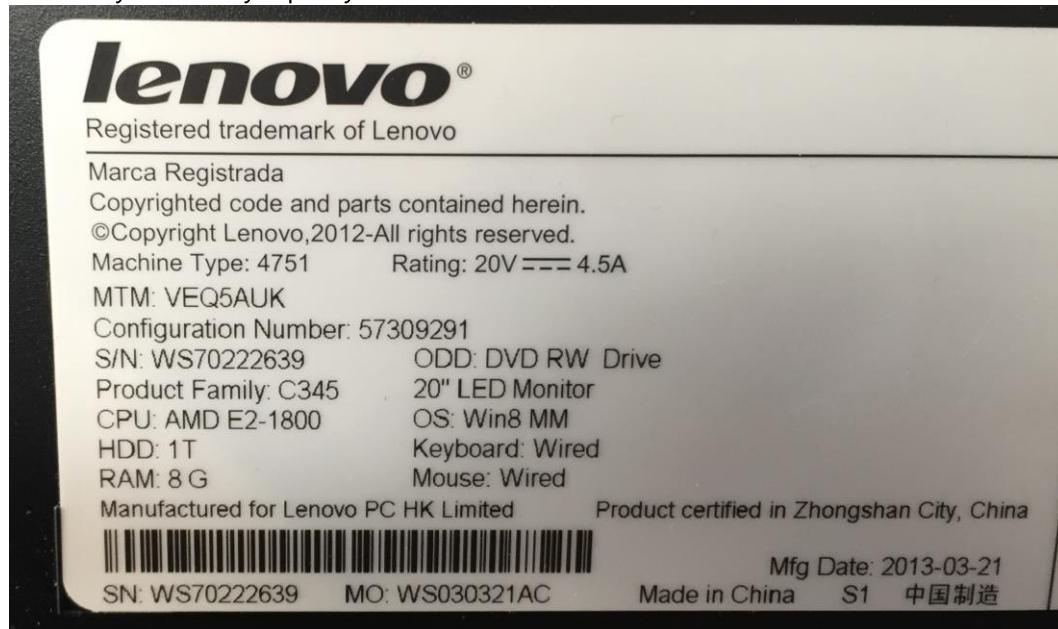
- It is a relatively new technology so may require additional caution if used.
- Needs to be determined whether the target computers will have the correct browsers installed.

### **Decision on web platform**

As the user has little background knowledge on web platforms I will be making a decision as to which platform the system will be developed with. This is because I have better knowledge in this area. I intend to use the HTML 5 Canvas element in order to display the game in a web browser. This will allow the game to be embedded, as a canvas element, in a webpage. I chose the canvas element because it does not require the use of proprietary technology and can be used to generate fast graphics in a web browser. It also allows me to develop the system with the JavaScript programming language.

## Investigating hardware and software requirements

The client, in the first interview, discussed the prospect of the game running on the computers at the sixth form. Therefore, I investigated the hardware and software specifications of the school computer's in order to determine the requirements the game will have in terms of hardware and software. Below is an image of the hardware specifications of the school computer system. This was a photograph taken of the back of the computer's in the computer area. Note that the hardware of these computer systems is in excess of what would be required to run the game. The 8GB of RAM, 1TB Hard disk drive and an AMD E2-1800 processor are more than acceptable to run the new system in any capacity.



Therefore, in order to specify a more precise requirements specification I decided to investigate the hardware requirements of the web browsers that support the HTML 5 canvas element. This will allow me to tailor the hardware requirements to the software requirements for the game. On the following are images of the requirements for the latest versions of various web browsers.

These specification areas will influence but not dictate the hardware required for the system. For example, Google Chrome requires 350MB of disk space but the new system will require no such amount. However 512MB of RAM would be an acceptable amount to run a game of the capacity of that of the new system. This information will be used to justify the hardware and software requirements in the Requirements Specification.

### Mozilla Firefox hardware requirements:

Can't use it? Try [this](#), only 32-bit builds of Firefox are supported.

#### Recommended Hardware

- Pentium 4 or newer processor that supports SSE2
- 512MB of RAM
- 200MB of hard drive space

### Google Chrome browser requirements:

#### System requirements

For optimal performance, we recommend the following system requirements:

	Windows requirements	Mac requirements	Linux requirements
<b>Operating system</b>	<ul style="list-style-type: none"> <li>• Windows XP* Service Pack 2+ *supported until April 2015 ↗</li> <li>• Windows Vista</li> <li>• Windows 7</li> <li>• Windows 8</li> </ul>	Mac OS X 10.6 or later	Ubuntu 12.04+ Debian 7+ OpenSUSE 12.2+ Fedora Linux 17
<b>Processor</b>	Intel Pentium 4 or later	Intel	Intel Pentium 4 or later
<b>Free disk space</b>	350 MB		
<b>RAM</b>	512 MB		

### Internet Explorer browser requirements:

#### Internet Explorer system requirements

##### Processor

- Computer with a 1 gigahertz (GHz) 32-bit (x86) or 64-bit (x64) processor

##### Operating system

- Windows Vista 32-bit with Service Pack 2 (SP2) or higher
- Windows Vista 64-bit with Service Pack 2 (SP2) or higher
- Windows 7 32-bit or higher
- Windows 7 64-bit or higher
- Windows Server 2008 32-bit with Service Pack 2 (SP2) or higher
- Windows Server 2008 64-bit with Service Pack 2 (SP2) or higher
- Windows Server 2008 R2 64-bit

##### Memory

- Windows Vista 32-bit with Service Pack 2 (SP2) or higher—512 MB
- Windows Vista 64-bit with Service Pack 2 (SP2) or higher—512 MB
- Windows 7 32-bit or higher—512 MB
- Windows 7 64-bit or higher—512 MB
- Windows Server 2008 32-bit with Service Pack 2 (SP2) or higher—512 MB
- Windows Server 2008 64-bit with Service Pack 2 (SP2) or higher—512 MB
- Windows Server 2008 R2 64-bit—512 MB

##### Hard drive space

- Windows Vista 32-bit with Service Pack 2 (SP2) or higher—70 MB
- Windows Vista 64-bit with Service Pack 2 (SP2) or higher—120 MB
- Windows 7 32-bit or higher—70 MB
- Windows 7 64-bit or higher—120 MB
- Windows Server 2008 32-bit with Service Pack 2 (SP2) or higher—150 MB
- Windows Server 2008 64-bit with Service Pack 2 (SP2) or higher—200 MB
- Windows Server 2008 R2 64-bit—200 MB

##### Drive

- CD-ROM drive (if installation is done from a CD-ROM)

##### Display

- Super VGA (800 x 600) or higher-resolution monitor with 256 colors

##### Peripherals

- Modem or Internet connection
- Microsoft Mouse, Microsoft IntelliMouse, or compatible pointing device

## Second interview

### Introduction

The first interview provided an overview of the theme and scope of the new system solution and lead to the need for further research into the problem area – including existing systems, existing algorithms, possible web technologies and possible hardware and software requirements. With this gathered it is necessary to undertake a second interview. Questions will be prepared that will probe further into the system's requirements in order to create a workable requirements specification the client will be happy to sign off on. It will take a conversational style akin to that of the first interview. Below some of the points that I will attempt to touch in the interview have been summarised.

### Discussion points in the interview

- The client's views on the user interface of the game need to be discussed.
- The client's views on the colours of game elements (tiles, walls, etc.) need to be discussed.
- The composition of the elements that could make up the user interface of the game (windows / popups / buttons etc.) needs to be discussed with the client.
- The client's views on the size of the game elements.
- The client's views on the saving and loading game feature.
- The client's views on features from existing systems (specifically the time taken to complete the maze / number of steps taken to complete the maze being recorded).
- The client's views, if any, on the presence of onscreen help.
- Discussion of the controls that can be used.
- Will the mazes be randomly generated or premade by a person.

### Transcript of the second interview

*Me:* Thank you for coming to this interview.

*Client:* No problem.

*Me:* I've been doing some online research after our first discussion. I had a look at some existing maze games and one of them displays the number of steps taken to complete the maze and the time taken at the end. I encourage that we include that. What do you think?

*Client:* I think it's a great idea. Definitely make that a feature.

*Me:* Now, this interview is really about getting as much information about features and how you want the game to look and feel as possible. So I'm going to just have a discussion with you about features and game visuals and see where it leads us. That okay with you?

*Client:* Yeah that's fine.

*Me:* Okay great. So have you had any ideas since the last meeting about the game?

*Client:* I've had some ideas myself that I want to talk to you about. I want to call the game Mazr [pronounced Mazer]. This is quite a modern style of naming games and apps I think. So I imagine the maze to be like a grid of tiles. Each of the tiles has walls that make up the maze. The player cant move through the walls or off the edge of the maze and so has to find the path through the tiles to the exit tile. This is a special tile and if you reach this tile then that maze is completed.

*Me:* Okay that's very interesting. What kind of controls will move the player?

*Client:* I think the arrow keys are the obvious choice so I'd go with that.

*Me:* Great I've made a note of that. Do you have any other ideas?

*Client:* I think that the game needs to be in a window in the website. Most games I've played are like this and it'd be more natural. The game needs to have menus the user can view.

*Me:* Okay, that's a good idea. So like a traditional menu system on a game where buttons allow you to navigate through different menus?

*Client:* Yeah exactly. There needs to be a main menu. The one that would be shown when you first go to the website. This menu needs to have buttons like new game, or load a saved game, etc.

*Me:* Okay, good idea. What about help? Will there be instructions?

*Client:* Oh yes! Have a button that goes to a menu that shows the help.

*Me:* Will all the help go on one menu? Remember it has to fit in the game window. Will it scroll down? Or will the help just be short?

*Client:* Hmm. I'm not sure really.

*Me:* It depends on how much help there'll be, maybe images as well to show how the instructions appear.

*Client:* I think we'd have a hard time fitting all the help on one menu. What else could you do, other than scrolling?

*Me:* Maybe have the help over a few menus that you can navigate through.

*Client:* Oh yes, that'd work well. Navigate with arrow buttons. Like how'd you navigate through photos in a photo viewer?

*Me:* So like two arrow buttons, one left and one right? Pressing right shows the next help menu and pressing left shows the previous help menu?

*Client:* Yeah that'd be good. Oh and a button to go back to the main menu. That could be on every help menu.

*Me:* Okay, I'll make a note of that. So back to the main menu. The new game button starts a new maze. The continue button loads a saved game. We can talk more about the game saving mechanic later. What other menus does there need to be do you think?

*Client:* So another one I had in mind was that there should be some sort of menu when you finish a maze. It could show the time needed to solve that maze and the number of steps the player took. And then have a button that loads the next maze.

*Me:* Yep, I'll make a note of that.

*Client:* Also, there could be a pause menu that stops the timer.

*Me:* How would the user navigate to the pause menu? Would it be a button or a key press?

*Client:* Most games use the escape key so I'd go with that.

*Me:* This is a feature that the existing maze games don't have and I think is a good idea. Would you want the escape key toggle the pause menu so you could use it to return to the maze?

*Client:* Yes that'd work well. There should be a button that goes back to the main menu on the pause menu. Oh and on the menu displayed at the end of every level.

*Me:* Okay I'll make a note of that. I think I have enough preliminary information on the menus. Let's talk now about the graphical style. How do you envision the maze to look?

*Client:* All of the mazes should be square. I like the idea of everything in the game being square or rectangular. Square maze with square tiles and square player and square exit. And rectangular walls and buttons.

*Me:* Okay. So let's talk more about the colour scheme. What colour should the tiles in the maze be?

*Client:* I first thought lets make them grey. But then I wanted to have them like a chessboard with alternating white and black. Then I was thinking that maybe a chessboard pattern was too distinct. I liked the idea of having a slight difference in colour in each tile. Like a brick wall with loads of different coloured bricks.

*Me:* Okay. So each tile could be a unique colour? What kind of colours?

*Client:* I was thinking all quite light shades of grey. So you can see each of the individual tiles but they all fit the same style.

*Me:* Sounds reasonable. What about the walls?

*Client:* I think the walls should all be dark grey to contrast a lighter grey floor.

*Me:* What about the background of the game - behind the maze or beyond the edges?

*Client:* A grey that is lighter than the walls but darker than the tiles. Do you understand what I mean?

*Me:* Okay, that's fine. So what about the player's colour?

*Client:* I've had an interesting thought about this. I was talking to a friend about the ideas I had for the graphics in the game and he said he thinks it'll be too grey. So I decided the player needs to be really vibrant and colourful to contrast.

*Me:* That makes sense. What did you have in mind?

*Client:* I think the player should fade through colours. Like slowly change colour over time. I think that would be interesting for the game. Would make good contrast with all the grey.

*Me:* What colours?

- Client:* I was thinking a pale green to start and go all the way through to a purple then back to pale green. Like a cycle. Do you know what I mean?
- Me:* Yes that's fine. It's an interesting idea and would a fun addition. What about the exit tile? What colour would that be?
- Client:* I was thinking red to contrast with the player's fading colours. But then I thought maybe red signifies warning or danger. So something less like a hazard sign. Maybe orange? It'd look good against the grey.
- Me:* I agree. What about the trail the player is supposed to leave. What colour should that be?
- Client:* Make it the same as the player but a bit lighter. So it matches the player and leaves a faded colour. But because it's lighter you can tell it's not the player.
- Me:* Okay, noted. Moving on I'd like to talk about the mechanics of the game, if that's okay with you?
- Client:* Yeah that's fine with me.
- Me:* Good. So we discussed in our first interview together that the game could make use of randomly generated mazes. I've been doing some research into this and I've found not only is it possible but quite easy to include by following a procedure that's already had lots of research put into it. Would you like to proceed with this method of creating the mazes?
- Client:* Yes I think that'd be the best way forward. It's almost like a USP [Unique Selling Point]. The random mazes make for a very cool feature.
- Me:* That's a good choice. I'm going to be using something called a randomised depth first search algorithm. I don't expect you to know anything about it but I wanted to let you know I'm using this method because it should be an easy way to produce the random mazes.
- Client:* Okay sounds fine with me. I was thinking that to make the game get more challenging as you go along the mazes should get bigger. So each maze is bigger than the last. To keep the game interesting.
- Me:* Good idea I like that. What size would the first maze be?
- Client:* I think it should start off very small, just so you can get used to the controls. I was thinking 4 by 4 tiles.
- Me:* Okay good idea. And what rate should the mazes grow? What does it increase by with each completed maze?
- Client:* It needs to be a slow increase otherwise the game could get too hard. I'm thinking 2 larger in each direction.
- Me:* So like the level after a 4 by 4 tile maze would be a 6 by 6 tile maze?
- Client:* Exactly! I think that'd be a good rate.

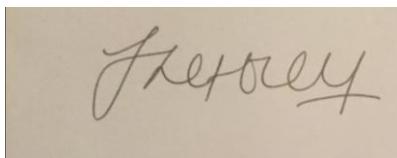
*Me:* Finally, I'd like to talk about the save and loading of the game. What needs to be saved?

*Client:* Um, so the game needs to be saved every time a maze is solved. And it needs to be the information about the next maze. This would include the level that the player is on. So a number value – level sixteen, for example. The size of that maze needs to be saved as well. So whether it's a 16 by 16 or 30 by 30 maze.

*Me:* Okay I'll make a note of that. I'd like to conclude this interview now. I think we've had a good lengthy discussion and I'll begin writing upon a requirements specification for you to review at our next meeting. Thank you very much.

### **Signature of client**

*The signature below certifies that the client agrees this is an accurate reproduction of the conversation that took place during the interview.*



### **Analysis of second interview**

With the second interview completed I now have a much firmer understand of how the final system will appear once it has been created. I decided to summarise the points we discussed. I will use these to formalise the user requirements of the Requirements specification that will be written later.

- The client wishes the game be called Mazr.
- The time and number of steps taken to complete a maze should be displayed to the player.
- Arrow keys control movement of player.
- Player cannot move through walls and off edge of the maze.
- A maze is considered solved when the player reaches the position of the 'exit tile' – special tile designated as the exit.
- The client thinks that the game must be embedded in a small window as this is what most other games do and the client finds this the most natural.
- The client thinks the game needs to display a main menu when webpage is first visited.
- This main menu needs a new game button that starts a new maze, a continue button that loads a saved maze and a help button that displays the onscreen help.
- The client wants onscreen help in the form of help menus that will be navigable by a series of arrow buttons. The right arrow buttons will display the subsequent help menu. The left arrow button will display the previous help menu.
- The client requested the presence of a pause menu reached and returned from by pressing the escape key. The game timer would be paused when on the pause menu.
- When each maze is completed, the client wants an end of level menu to be displayed that shows the steps taken and time taken to complete the previous maze. This menu should contain a button that generates the next maze.
- Both the pause menus and escape menus should have buttons that go back to the main menu.
- The client wishes that mazes be composed of a square grid of square tiles.
- The tiles in the mazes should be a random assortment of shades of grey.
- Walls should be a very dark grey.
- Background should be a dark grey.
- The player should fade through a spectrum of colours over time.
- The exit tile should be orange.
- The player needs to leave a trail in the same colour as itself. This colour needs to be slightly lighter so that it can be differentiated from the player.

- Mazes will be randomly generated. I decided to use a randomised depth first search algorithm, as this should be the easiest to implement and yield good variation in mazes. I made the client aware this is the path I will take with the maze generation.
- First maze is 4 by 4 tiles in horizontal and vertical dimensions. Each subsequent maze is 2 tiles larger in each dimension than the last.
- The level and dimensions of the next maze need to be saved every time a maze is completed.

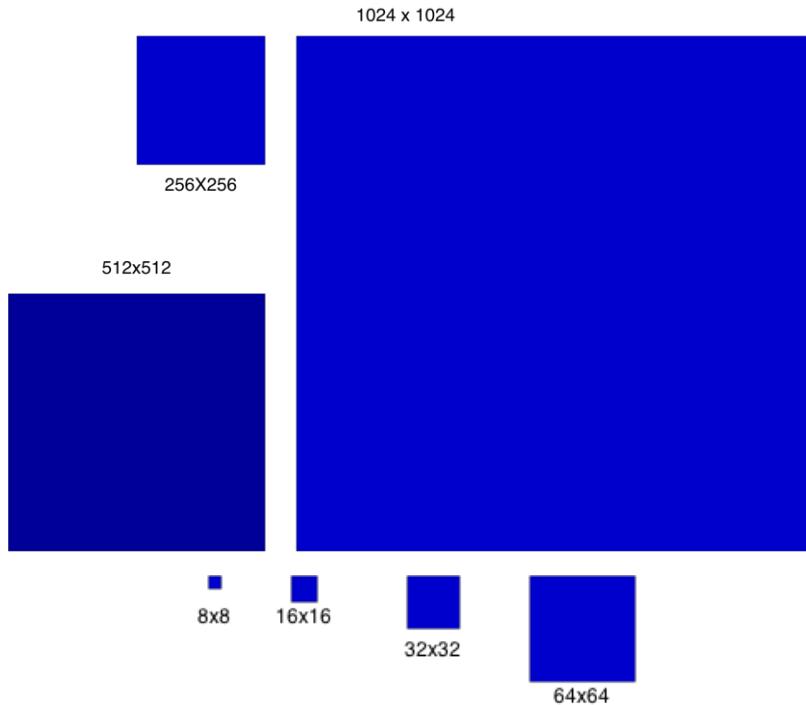
There are still some areas of the problem that need to be discussed. These are summarised below.

- The size of all of the game elements (the window, the tiles) needs to be discussed. This will take place in a very short third interview with the client.

On reflection of this interview, it became apparent there is a dense amount of information extracted from the conversation that took place in this interview. It will be unsuitable to include all of it in the requirements specification, which should be kept short at this stage in order to improve the ease of evaluating it at the end of the project. However, this information will be very useful in the Nature of the solution section that will be completed after this. The design objectives, a complete list of objectives pertaining to the whole system, will be a perfect opportunity to include a lot of this information. Therefore, much of it will be retained as research for the Design stage.

## Third interview

The purpose of the third interview is for the client to decide on the size they wish for each game element to be. In order to convey examples of the different sized elements that can be used I decided to produce the following diagrams below. These were shown to the client on a computer screen at the correct resolutions. Note the images reproduced below are not true to the actual size.



### Third interview transcript

*Me:* Thank you for coming. This will be a very short interview, as I just have to ask you a few quick questions. These are all about what size the elements of the game will be.

*Client:* Okay that's fine with me.

*Me:* Excellent. Well first I want you to look at these diagrams I've produced. They show rectangles at different resolutions.

[At this point in the interview I presented the diagrams to the client on my laptop]

*Me:* As you can see there are a variety of sizes. Firstly, what size do you want the game window and menus to be?

*Client:* A 512 by 512 pixel seems like a good size. I think we should go with that.

*Me:* Okay, what about the tiles in the maze? This includes the floor and the player and the exit tile

*Client:* 32 by 32 pixels? That would be quite a large maze that could fit in the window.

*Me:* Yeah, it'd be a 16 by 16 tile maze.

*Client:* What will happen when they finish that maze?

*Me:* That's up to you. You could have controls for moving the part of the maze being displayed? Or a camera that follows the player around the maze?

*Client:* A camera that follows the player would be perfect I think.

*Me:* Okay. So do you want the camera to always follow the player or only when the maze is rendered greater than 512 by 512 pixels?

*Client:* Only when the maze exceeds the size of the game window.

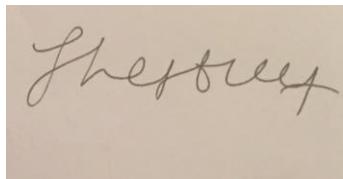
*Me:* What about when the maze does fit in the game window?

*Client:* Turn the camera off and centre the maze in the game window.

*Me:* Okay, I'll make a note of that. Thank you.

### **Signature of client**

*The signature below certifies that the client agrees this is an accurate reproduction of the conversation that took place during the interview.*



### **Analysis of third interview**

This was a short interview and as such has only a small amount of analysis that needs to be completed. Below are the conclusions of the third interview:

- The tiles in the maze need to be 32 pixels by 32 pixels.
- The game window that contains the canvas and the menus needs to be 512 by 512 pixels.
- A camera needs to follow the player around on mazes where the whole of the maze does not fit in the canvas.
- The maze needs to be fixed in the centre of the canvas when it fits inside the dimensions of the canvas.

With this interview completed I now had all of the necessary information in order to produce a draft requirements specification for the user to review.

## Draft Requirements specification

This section contains a draft copy of the various requirements the system has been determined to have at this stage in the analysis process. These are divided into three subsections – user, software and hardware. The client will review this draft copy in order to ensure it meets their preferred requirements.

User requirements are a list of requirements agreed to with the client deemed to be suitably appropriate to the end user. The software requirements are all of the points that pertain to the software required to run the game. Hardware requirements are all the points that pertain to the hardware required to run the game. Hardware and software requirements are justified to reflect the research that went into deciding them. The user requirements were ascertained through careful analysis of the interviews that took place and any further research that was completed using the Internet.

## User requirements

### **Input Requirements**

1. Clicking a menu button with the mouse pointer will navigate to another page.
2. Pressing the arrow keys when displaying a maze will allow the player to move by one tile in one direction dependent on the specific arrow key.
3. Pressing the WASD when displaying a maze will allow the player to move by one tile in one direction dependent on the specific letter key.
4. Pressing the escape key when a maze is displayed will pause the game. Pressing the escape key when the game is paused will resume the game.
5. Pressing the new game button starts a new 4 by 4 tile maze.
6. Pressing the continue button should load game data from local storage and start a game of the correct size. When there is no game data saved then the continue button is disabled.
7. Pressing the help button navigates to the first help menu.
8. Pressing the arrow buttons that aren't disabled on the help pages allows user to navigate between help menus.
9. Pressing the back button on any help menu should navigate the user back to the main menu.

### **Processing Requirements**

10. New mazes are randomly generated with a randomised depth first search algorithm.
11. When a maze is generated it must be ensured the player tile and exit tile are separated by a distance of at least a third of the size of the side of a maze.
12. Input from the WASD keys must be used to determine player movement. A wall must not block the path the player wishes to take.
13. The player must not be able to move off of the edges of the maze.
14. When the player reaches the exit tile then the maze is considered completed. When a maze is completed the 'end of level menu' should be displayed.
15. Every time the player completes a maze, the next maze they play will be 2 tiles larger in both dimensions (E.g. by completing a 4 by 4 tile maze the next maze will be 6 by 6 tiles).
16. When a maze is completed and an 'end of level menu' is displayed and game data should be saved so the player can continue later.
17. Game data must be loaded from local storage when the continue button is pressed.
18. The camera must follow the player in mazes where the maze is larger than the game window.

### **Output / Visual Requirements**

19. The game will be embedded as a HTML 5 canvas element in a HTML webpage, styled with CSS 3.
20. The main menu must be displayed when the user navigates to the webpage.
21. The main menu must contain three buttons – a continue button, a new game button and a help button.

22. There should be three help pages. Each has two buttons labelled with arrows (that can be used to navigate between the help pages) and a back button.
23. Each help page must contain text and images that assist in explaining and understanding the game.
24. The end of level menu should display the time taken for the player to complete the maze, the number of steps the player took, a continue button and a quit to main menu button.
25. The pause menu should contain a quit to main menu button.
26. The game window that is embedded within the web page should be 512 by 512 pixels.
27. The maze should start with 4 by 4 tiles.
28. The player must fade between varieties of colours.
29. The exit tile should be coloured orange.
30. The walls of the maze should be coloured dark grey.
31. The colour of the floor of the maze should be coloured light grey, random assorted pattern of shades of grey.
32. The game's camera must follow the player around the maze, with the player at the centre of the camera unless the maze is smaller than the viewport in which case the maze should remain centred in the camera.
33. When a new maze loads that is larger than what can be displayed in the window then the camera should show a preview of the exit tile and pan over to the player (*in order to give the player a general direction in which to head for the exit tile*).

#### Software requirements

Specification	Justification
The user must have one of the following operating systems installed and operational on their computer <ul style="list-style-type: none"> <li>• Windows 7 or later</li> <li>• Mac OS X Lion or later</li> </ul>	Windows 7 is the operating system that runs on the sixth form computers. Mac OS X Lion is being used on the computer that is being used to develop software and is the Apple equivalent on Windows 7. Therefore these have been chosen as the minimum Operating System versions for the system.
The user must use one of the following browsers to access the game: <ul style="list-style-type: none"> <li>• Google Chrome versions 27 and later</li> <li>• Internet Explorer versions 9 and later</li> <li>• Mozilla Firefox versions 30 and later</li> <li>• Safari versions 5.1 and later</li> </ul>	The game will be hosted on a web page so it is necessary for a browser to be used in order to access the game.  These browsers are all identified as being able to run HTML 5 canvas elements, upon which the game will be rendered.
The user must have JavaScript enabled on their browser.	All of the game logic will be programmed in JavaScript. Therefore it is necessary for the user to have their JavaScript enabled on the browser in which they access the game. Note that the browsers that the game targets have JavaScript engines that will run the game logic. Therefore, this is no explicit software requirement to have an up-to-date JavaScript engine.

#### Hardware requirements

Specification	Justification
The user must have an Internet connection.	The game will be encapsulated within a HTML web page that will be served over the WWW. Therefore, any hardware necessary to facilitate an internet connection is required to request the web page. This involves the requirement for any hardware that facilitates a network including, for example routers, bridges, switches and modems.
The user must have	This is because the HTML 5 local storage API allows up to 5MB of data to

5MB of storage space on their computer.	<p>be saved locally to a client computer by the browser.</p> <p>The game will use this storage to store data about mazes so it is determined that up to the maximum amount allocated to the game may be used. Note that the server stores no user data so is instead stored locally on the client's computer.</p> <p>Note while the browsers quote needing more space in their system requirements – this is not a requirement that transfers over to this game. The storage is used by the browser, which allocates files to that space. The game simply will simply manipulate a hash table of key value pairs in local storage that will not use more than maximum 5MB of data.</p>
The user must have a keyboard connected to their computer.	This is to capture input data from the user in order to move the player around a maze.
The user must have a mouse / touchpad connected to their computer.	This is so that the user can point and click on buttons on the menu. These are considered necessary hardware for all personal computer systems including desktops and laptops.
User must have at least a 1024x768 or equivalent resolution monitor connected to their computer.	<p>This is so they can view the web browser on which the game is being played. The game is being rendered in a window 512 pixels by 512 pixels so further space needs to be allowed to display the rest of the operating system features (such as taskbar and menu bar), the browser window and elements of the web page.</p> <p>1024x768 was chosen, although this is larger than the 800x600 resolution demanded by the system requirements for Internet Explorer, however this would lead to a poor rendering of the game's web page.</p>
The user must have 512MB of RAM	The minimum amount of RAM required to run the web browsers required to access the game's website.
The user must have 1GHz or faster CPU with 32 bit or 64 bit architecture.	This is a minimum value for the clock speed of the CPU. Although the clock speed is only one of many factors that affect the processing speed of a CPU, it is often an easily measurable factor that is easy to find out by a user. A 1GHz processor is required to run Internet Explorer and Google Chrome and Mozilla quote Pentium 4 processors, which are old processors that have been superseded by processors that can have clock speeds roughly as low as 1GHz.

**Review of requirements specification by client**

I decided to arrange an interview with the client to review the draft requirements statement produced previously. I supplied the client with a copy of the draft requirements specification in order for them to review it and suggest comments or modifications.

**Transcript of review of requirements specification**

Me: So now you've had a chance to review the requirements specification that I produced, do you have any comments or modifications to suggest?

Client: Yes. I wanted to mention that you've forgotten to add anything about the coloured trail being left by the player?

Me: Oh yes it would seem I have. I'll make a note to add a review about the trail colour.

Client: Yes. I was thinking about how some people prefer to play games with the WASD keys, so could I change requirement 3 so that the user can use the WASD or the arrow keys to move the player around the maze? Would that be okay to add?

Me: Yes of course I'll make an amendment to the requirements specification.

Client: Thank you.

Me: Is there anything else you'd like to add?

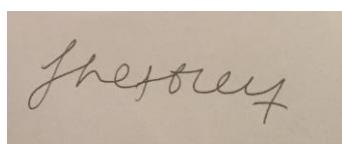
Client: No that's all I had to mention, thank you.

Me: Are you sure? No comments about the hardware and software requirements? Nothing I can clarify?

Client: No it all seems fine to me. I had to check up on the definition of a megabyte online but I know how much that is now. Thank you.

**Signature of client**

*The signature below certifies that the client agrees this is an accurate reproduction of the conversation that took place during the discussion.*

**Amended requirements specification**

The following pages contain the final requirements specification along with the corrections that have been made with respect to the user's comments in the review. The amended points are shown highlighted.

## Final Requirements specification

This section contains the various requirements the system has been determined to have at this stage in the system analysis process. These are divided into three subsections – user, software and hardware.

User requirements are a list of requirements agreed to with the client deemed to be suitably appropriate to the end user. The software requirements are all of the points that pertain to the software required to run the game. Hardware requirements are all the points that pertain to the hardware required to run the game. Each is organised into a headed table, all three of which follow.

## User Requirements

### Input Requirements

1. Clicking a menu button with the mouse pointer will navigate to another page.
2. Pressing the arrow keys when displaying a maze will allow the player to move by one tile in one direction dependent on the specific arrow key.
3. Pressing the WASD keys or the arrow keys when displaying a maze will allow the player to move by one tile in one direction dependent on the specific letter key.
4. Pressing the escape key when a maze is displayed will pause the game. Pressing the escape key when the game is paused will resume the game.
5. Pressing the new game button starts a new 4 by 4 tile maze.
6. Pressing the continue button should load game data from local storage and start a game of the correct size. When there is no game data saved then the continue button is disabled.
7. Pressing the help button navigates to the first help menu.
8. Pressing the arrow buttons that aren't disabled on the help pages allows user to navigate between help menus.
9. Pressing the back button on any help menu should navigate the user back to the main menu.

### Processing Requirements

10. New mazes are randomly generated with a randomised depth first search algorithm.
11. When a maze is generated it must be ensured the player tile and exit tile are separated by a distance of at least a third of the size of the side of a maze.
12. Input from the WASD keys must be used to determine player movement. A wall must not block the path the player wishes to take.
13. The player must not be able to move off of the edges of the maze.
14. When the player reaches the exit tile then the maze is considered completed. When a maze is completed the 'end of level menu' should be displayed.
15. Every time the player completes a maze, the next maze they play will be 2 tiles larger in both dimensions (E.g. by completing a 4 by 4 tile maze the next maze will be 6 by 6 tiles).
16. When a maze is completed and an 'end of level menu' is displayed and game data should be saved so the player can continue later.
17. Game data must be loaded from local storage when the continue button is pressed.
18. The camera must follow the player in mazes where the maze is larger than the game window.

### Output / Visual Requirements

19. The game will be embedded as a HTML 5 canvas element in a HTML webpage, styled with CSS 3.
20. The main menu must be displayed when the user navigates to the webpage.
21. The main menu must contain three buttons – a continue button, a new game button and a help button.
22. There should be three help pages. Each has two buttons labelled with arrows (that can be used to navigate between the help pages) and a back button.

23. Each help page must contain text and images that assist in explaining and understanding the game.
24. The end of level menu should display the time taken for the player to complete the maze, the number of steps the player took, a continue button and a quit to main menu button.
25. The pause menu should contain a quit to main menu button.
26. The game window that is embedded within the web page should be 512 by 512 pixels.
27. The maze should start with 4 by 4 tiles.
28. The player must fade between varieties of colours.
29. The player must leave a trail on the floor of the tiles of the maze that is a lighter shade of the colour of the player.
30. The exit tile should be coloured orange.
31. The walls of the maze should be coloured dark grey.
32. The colour of the floor of the maze should be coloured light grey, random assorted pattern of shades of grey.
33. The game's camera must follow the player around the maze, with the player at the centre of the camera unless the maze is smaller than the viewport in which case the maze should remain centred in the camera.
34. When a new maze loads that is larger than what can be displayed in the window then the camera should show a preview of the exit tile and pan over to the player (*in order to give the player a general direction in which to head for the exit tile*).

#### Software requirements

Specification	Justification
The user must use have one of the following operating systems installed and operational on their computer <ul style="list-style-type: none"> <li>• Windows 7 or later</li> <li>• Mac OS X Lion or later</li> </ul>	Windows 7 is the operating system that runs on the sixth form computers. Mac OS X Lion is being used on the computer that is being used to develop software and is the Apple equivalent on Windows 7. Therefore these have been chosen as the minimum Operating System versions for the system.
The user must use one of the following browsers to access the game: <ul style="list-style-type: none"> <li>• Google Chrome versions 27 and later</li> <li>• Internet Explorer versions 9 and later</li> <li>• Mozilla Firefox versions 30 and later</li> <li>• Safari versions 5.1 and later</li> </ul>	The game will be hosted on a web page so it is necessary for a browser to be used in order to access the game.  These browsers are all identified as being able to run HTML 5 canvas elements, upon which the game will be rendered.
The user must have JavaScript enabled on their browser.	All of the game logic will be programmed in JavaScript. Therefore it is necessary for the user to have their JavaScript enabled on the browser in which they access the game. Note that the browsers that the game targets have JavaScript engines that will run the game logic. Therefore, this is no explicit software requirement to have an up-to-date JavaScript engine.

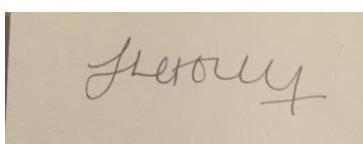
#### Hardware requirements

Specification	Justification
The user must have an Internet connection.	The game will be encapsulated within a HTML web page that will be served over the WWW. Therefore, any hardware necessary to facilitate an internet connection is required to request the web page. This involves the requirement for any hardware that facilitates a network including, for example routers, bridges, switches and modems.
The user must have	This is because the HTML 5 local storage API allows up to 5MB of data to

5MB of storage space on their computer.	<p>be saved locally to a client computer by the browser.</p> <p>The game will use this storage to store data about mazes so it is determined that up to the maximum amount allocated to the game may be used. Note that the server stores no user data so is instead stored locally on the client's computer.</p> <p>Note while the browsers quote needing more space in their system requirements – this is not a requirement that transfers over to this game. The storage is used by the browser, which allocates files to that space. The game simply will simply manipulate a hash table of key value pairs in local storage that will not use more than maximum 5MB of data.</p>
The user must have a keyboard connected to their computer.	This is to capture input data from the user in order to move the player around a maze.
The user must have a mouse / touchpad connected to their computer.	This is so that the user can point and click on buttons on the menu. These are considered necessary hardware for all personal computer systems including desktops and laptops.
User must have at least a 1024x768 or equivalent resolution monitor connected to their computer.	<p>This is so they can view the web browser on which the game is being played. The game is being rendered in a window 512 pixels by 512 pixels so further space needs to be allowed to display the rest of the operating system features (such as taskbar and menu bar), the browser window and elements of the web page.</p> <p>1024x768 was chosen, although this is larger than the 800x600 resolution demanded by the system requirements for Internet Explorer, however this would lead to a poor rendering of the game's web page.</p>
The user must have 512MB of RAM	The minimum amount of RAM required to run the web browsers required to access the game's website.
The user must have 1GHz or faster CPU with 32 bit or 64 bit architecture.	This is a minimum value for the clock speed of the CPU. Although the clock speed is only one of many factors that affect the processing speed of a CPU, it is often an easily measurable factor that is easy to find out by a user. A 1GHz processor is required to run Internet Explorer and Google Chrome and Mozilla quote Pentium 4 processors, which are old processors that have been superseded by processors that can have clock speeds roughly as low as 1GHz.

#### **Client's agreement to requirement's specification**

*The signature below certifies that the client has agreed to the requirement's specification – including the user specification pertaining to input, processing and output and the hardware and software requirements.*



# Nature of the solution

## Draft design objectives

The design objectives details all the objectives that must be achieved to ensure the game is designed with respect to the client's requirements and thus the requirement specification. The objectives are split into input, processing and output objectives. Each of these is provided with a unique identifier (e.g. 1.a.i) so they can be referenced later in the report. This will be reviewed by the client and modified based upon her views and opinions. Note there may be some overlap between objectives, as they will include a possible combination of input, processing and output facilities. Any particularly crucial information contained is bolded and any examples contained are italicised.

### 1) Input Objectives

- a) When on a menu (the main menu, help menus, end of level menu and pause menu):
  - i) When the **new game button** is pressed a new maze must be generated and displayed.
  - ii) When the **continue button** is **not disabled** and pressed then saved data must be loaded from Local Storage and used to generate a maze of the saved numerical level and with the dimensions of the maze the player saved on.
  - iii) Pressing the **help button** on the main menu will navigate the user to the first help menu.
  - iv) Pressing the **back button** on any of the three help menus will navigate the user back to the main menu.
  - v) Pressing the **right arrow button** on any of the three help menus, where it is **not disabled**, will navigate the user to the next help menu. *For example, pressing the right arrow button on the first help menu will navigate the user to the second help.*
  - vi) Pressing the **left arrow button** on any of the three help menus, when it is **not disabled**, will navigate the user to the previous help menu. *For example, pressing the left arrow button on the second help menu will navigate the user to the first help menu.*
  - vii) Pressing the **quit to main menu button** on the pause menu or on the end of level menu will navigate the user to the main menu.
  - viii) Pressing the **next level button** on the end of level menu generates and displays a new maze larger by 2 tiles in the horizontal and vertical directional and with an incremented level value.
- b) When solving a maze (not navigating a menu), the user can use the following controls:
  - i) Pressing the '**W**' **key** will cause the player to move up by one tile in the maze, unless a wall blocks them.
  - ii) Pressing the **up arrow key** will cause the player to move up by one tile in the maze, unless a wall blocks them.
  - iii) Pressing the '**A**' **key** will cause the player to move left by one tile in the maze, unless a wall blocks them.
  - iv) Pressing the **left arrow key** will cause the player to move left by one tile in the maze, unless a wall blocks them.
  - v) Pressing the '**S**' **key** will cause the player to move down by one tile in the maze, unless a wall blocks them.
  - vi) Pressing the **down arrow key** will cause the player to move down by one tile in the maze, unless a wall blocks them.
  - vii) Pressing the '**D**' **key** will cause the player to move right by one tile in the maze, unless a wall blocks them.
  - viii) Pressing the **right arrow key** will cause the player to move right by one tile in the maze, unless a wall blocks them.
  - ix) Pressing the **escape key** while a maze is displayed will cause the game to be paused and the pause menu to be displayed. If the pause menu is currently being displayed then pressing the escape key on the keyboard will cause the user to return to the maze. The escape key does nothing when on any other menu.

**2) Processing Objectives**

- a) When a new maze is being generated and displayed:
  - i) The maze should be generated procedurally with a randomised depth first search algorithm. All mazes must have a solution.
  - ii) The first level maze should be 4 by 4 tiles in size.
  - iii) The player and exit zone are randomly placed in the maze at least a third of the size of the maze apart.
  - iv) If the maze does not fit in the canvas (512 by 512 pixels) then the camera should be enabled and follow the player around the maze
  - v) If the camera is enabled, when the new maze loads and displays for the first time, the camera should be centred on the exit zone and slowly pan back towards the centre of the player. This is so that the player gets a view of the maze's exit before they can begin solving it.
- b) Each subsequent maze should be 2 tiles larger than the previous maze in the horizontal and vertical directions. *For example, if the 4 by 4 maze is completed then the next maze displayed should be 6 by 6 tiles.*
- c) When there is no saved game data in Local Storage the continue button should be disabled.
- d) When the game is being played:
  - i) The player should not move if a movement command is entered and a wall blocks the direction in which the player is attempting to move.
  - ii) The player should not move if a movement command is entered and the edge of the maze blocks the direction in which the player is attempting to move.
  - iii) The player should not move if the player is disabled. The player is disabled when menus are being displayed and when the camera is panning from the exit tile to the player (this occurs on mazes that cannot fill the canvas).
  - iv) Mark the maze as completed and load the end of level menu if the position of the player and the exit tile is the same.
  - v) As the player moves over a tile, it should colour the tile with a modified colour of the player – maintaining the same hue but with a 20% increase in lightness and 10% decrease in saturation.
  - vi) The colour of the player tile should fade through a spectrum of hues slowly over time. The player should fade from green to purple and then back again in a recurring cycle.
  - vii) Update the position of the player every 0.01 seconds in small increments to ensure the player moves smoothly.
  - viii) Update the position of the camera every 0.01 seconds.

**3) Output Objectives**

- a) The game will be embedded in a single web page.
- b) This web page will contain:
  - i) A header with the name of the game "Mazr" should be displayed.
  - ii) A subheading with the text "The endless maze solving puzzle game" should be displayed.
  - iii) The game should be embedded in a HTML canvas element.
- c) If JavaScript is disabled in the user's browser then no menus or the canvas should be displayed and an error message with the text "You need to enable JavaScript in your browser in order to play the game" should be displayed.
- d) The game will be composed of a set of *menus*, containers within the webpage that hold information. Menus will be contained in div HTML elements and all appear on the same page although hidden / unhidden to simulate multiple pages being navigated.
- e) The game's *main menu* will contain:
  - i) A continue button, to resume a level saved in local storage. This is disabled if no game is stored in local storage.
  - ii) A new game button that starts a new game.

- iii) A help button to go to the first help menu.
- f) The game will contain 3 *help menus*. Each one will contain:
  - i) A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.
  - ii) A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.
  - iii) A back button to return to the main menu from any of the help menus.
- g) The first help menu will also contain:
  - i) Onscreen help that explains the format of the game and the goal of the game.
  - ii) An image of a small maze with a player and exit tile.
- h) The second help menu will also contain:
  - i) Onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.
  - ii) Image of a small maze where the player is leaving a coloured trail moving to the exit tile.
- i) The third help menu will also contain:
  - i) Onscreen help that explains how completing a maze subsequently generates a larger maze for the player to solve.
  - ii) Image of a larger maze.
- j) The *end of level menu* should contain:
  - i) Text congratulating the user on their completion of the maze.
  - ii) Text that contains the time taken for the user to complete the maze.
  - iii) Text that contains the number of steps taken for the user to complete the maze.
  - iv) The next level button, which allows the user to proceed to a newly generated maze.
  - v) The quit to main menu button, a button that allows the user to quit to the main menu.
- k) The *pause menu* should contain:
  - i) The quit to main menu button, a button that allows the user to quit to the main menu.
  - ii) Small amount of text expressing that pressing the escape key will return the user to the maze.
- l) When the game is executing and a maze being displayed:
  - i) All of the game elements (such as tiles, the maze, player, exit tile, etc.) will be drawn to the screen with a HTML 5 canvas element.
  - ii) The canvas should be encapsulated within a 512 by 512 pixel bordered container.
  - iii) A grid of tiles with up to four walls on each side that represents the maze must be drawn to the canvas, along with a player and exit tile.
  - iv) The tiles in the game should be 32 by 32 pixels.
  - v) The player tile should be 32 by 32 pixels.
  - vi) The exit tile should be 32 by 32 pixels.
  - vii) The colours of the tiles in the maze (excluding the player and exit tile) should be assorted random shades of light grey.
  - viii) Saturation and lightness of player tile should remain constant at values of 80% and 40% respectively.
  - ix) The colour of the exit zone should be orange with hex value #FF8000.
  - x) If the maze fits in the canvas (512 by 512 pixels) then the maze should be centred in the canvas and the camera disabled.
  - xi) If the maze doesn't fit in the dimensions of the canvas (512 by 512 pixels) then a camera should be enabled that has the player positioned at the centre and follows the centre of the player, after panning has occurred.
  - xii) The hue of the player must constantly change between the values 145 to 300.
  - xiii) The player's trail must be displayed. Each tile visited by the player must be coloured with a lighter shade of the colour of the player.

**Agreement to design objectives with client**

The client was provided with a printed copy of the draft design objectives to review over the course of a few days and they were generally happy with the objectives that were put in place. Below is a transcript of the conversation that took place between the client and I.

*Me:* Now you've had a chance to read the design objectives, are you happy with how it looks and are there any points you'd like to discuss being changed.

*Client:* Yes, *I noticed* there's no mention about the size of the walls. I realised we never discussed it in the interviews. Is there something we can do about that?

*Me:* Yes, that's a very good point and of course we can work it out. Do you have a size in mind or shall I produce some mock-up diagrams and we can reschedule this discussion?

*Client:* Well I think I can just relate it back to the size of the tiles. If they're 32 by 32 pixels then the length of the walls must be 32 pixels, right? As in the long side of the walls. And I want the walls to be very thin. Would 2 pixels be appropriate?

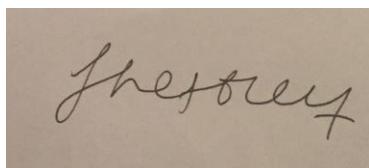
*Me:* That sounds good to me. I'll add that to the output design objectives. Do you have any other comments?

*Client:* You could add a fourth objective under [3.b]. A link to some sort of manual they could download would be good.

*Me:* I will be producing a PDF that contains user documentation. A link to this on the website would be a very good idea. I will add that to the final design objectives.

**Signature of client**

*The signature below certifies that the client agrees this is an accurate reproduction of the conversation that took place during the discussion of the design objectives.*

**Final design objectives**

The design objectives have therefore been modified to reflect the client's changes. The new design specification is reproduced overleaf and the modifications are shown highlighted and underlined.

## Final design objectives

### 1) Input Objectives

- a) When on a menu (the main menu, help menus, end of level menu and pause menu):
  - i) When the **new game button** is pressed a new maze must be generated and displayed.
  - ii) When the **continue button** is **not disabled** and pressed then saved data must be loaded from Local Storage and used to generate a maze of the saved numerical level and with the dimensions of the maze the player saved on.
  - iii) Pressing the **help button** on the main menu will navigate the user to the first help menu.
  - iv) Pressing the **back button** on any of the three help menus will navigate the user back to the main menu.
  - v) Pressing the **right arrow button** on any of the three help menus, where it is **not disabled**, will navigate the user to the next help menu. *For example, pressing the right arrow button on the first help menu will navigate the user to the second help.*
  - vi) Pressing the **left arrow button** on any of the three help menus, when it is **not disabled**, will navigate the user to the previous help menu. *For example, pressing the left arrow button on the second help menu will navigate the user to the first help menu.*
  - vii) Pressing the **quit to main menu button** on the pause menu or on the end of level menu will navigate the user to the main menu.
  - viii) Pressing the **next level button** on the end of level menu generates and displays a new maze larger by 2 tiles in the horizontal and vertical directional and with an incremented level value.
- b) When solving a maze (not navigating a menu), the user can use the following controls:
  - i) Pressing the '**W**' **key** will cause the player to move up by one tile in the maze, unless a wall blocks them.
  - ii) Pressing the **up arrow key** will cause the player to move up by one tile in the maze, unless a wall blocks them.
  - iii) Pressing the '**A**' **key** will cause the player to move left by one tile in the maze, unless a wall blocks them.
  - iv) Pressing the **left arrow key** will cause the player to move left by one tile in the maze, unless a wall blocks them.
  - v) Pressing the '**S**' **key** will cause the player to move down by one tile in the maze, unless a wall blocks them.
  - vi) Pressing the **down arrow key** will cause the player to move down by one tile in the maze, unless a wall blocks them.
  - vii) Pressing the '**D**' **key** will cause the player to move right by one tile in the maze, unless a wall blocks them.
  - viii) Pressing the **right arrow key** will cause the player to move right by one tile in the maze, unless a wall blocks them.
  - ix) Pressing the **escape key** while a maze is displayed will cause the game to be paused and the pause menu to be displayed. If the pause menu is currently being displayed then pressing the escape key on the keyboard will cause the user to return to the maze. The escape key does nothing when on any other menu.

### 2) Processing Objectives

- a) When a new maze is being generated and displayed:
  - i) The maze should be generated procedurally with a randomised depth first search algorithm. All mazes must have a solution.
  - ii) The first level maze should be 4 by 4 tiles in size.
  - iii) The player and exit zone are randomly placed in the maze at least a third of the size of the maze apart.
  - iv) If the maze does not fit in the canvas (512 by 512 pixels) then the camera should be enabled and follow the player around the maze

- v) If the camera is enabled, when the new maze loads and displays for the first time, the camera should be centred on the exit zone and slowly pan back towards the centre of the player. This is so that the player gets a view of the maze's exit before they can begin solving it.
- b) Each subsequent maze should be 2 tiles larger than the previous maze in the horizontal and vertical directions. *For example, if the 4 by 4 maze is completed then the next maze displayed should be 6 by 6 tiles.*
- c) When there is no saved game data in Local Storage the continue button should be disabled.
- d) When the game is being played:
  - i) The player should not move if a movement command is entered and a wall blocks the direction in which the player is attempting to move.
  - ii) The player should not move if a movement command is entered and the edge of the maze blocks the direction in which the player is attempting to move.
  - iii) The player should not move if the player is disabled. The player is disabled when menus are being displayed and when the camera is panning from the exit tile to the player (this occurs on mazes that cannot fill the canvas).
  - iv) Mark the maze as completed and load the end of level menu if the position of the player and the exit tile is the same.
  - v) As the player moves over a tile, it should colour the tile with a modified colour of the player – maintaining the same hue but with a 20% increase in lightness and 10% decrease in saturation.
  - vi) The colour of the player tile should fade through a spectrum of hues slowly over time. The player should fade from hue values 145 to 300 in the HSL colour range.

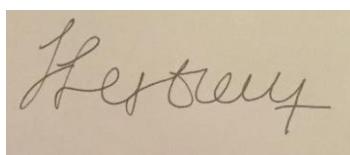
### 3) Output Objectives

- a) The game will be embedded in a single web page.
- b) This web page will contain:
  - i) A header with the name of the game "Mazr" should be displayed.
  - ii) A subheading with the text "The endless maze solving puzzle game" should be displayed.
  - iii) The game should be embedded in a HTML canvas element.
  - iv) A footer that contains a link to the user documentation, which will be provided as a PDF document, should be displayed.
- c) If JavaScript is disabled in the user's browser then no menus or the canvas should be displayed and an error message with the text "You need to enable JavaScript in your browser in order to play the game" should be displayed.
- d) The game will be composed of a set of *menus*, containers within the webpage that hold information. Menus will be contained in div HTML elements and all appear on the same page although hidden / unhidden to simulate multiple pages being navigated.
- e) The game's *main menu* will contain:
  - i) A continue button, to resume a level saved in local storage. This is disabled if no game is stored in local storage.
  - ii) A new game button that starts a new game.
  - iii) A help button to go to the first help menu.
- f) The game will contain 3 *help menus*. Each one will contain:
  - i) A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.
  - ii) A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.
  - iii) A back button to return to the main menu from any of the help menus.
- g) The first help menu will also contain:
  - i) Onscreen help that explains the format of the game and the goal of the game.
  - ii) An image of a small maze with a player and exit tile.
- h) The second help menu will also contain:

- i) Onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.
- ii) Image of a small maze where the player is leaving a coloured trail moving to the exit tile.
- i) The third help menu will also contain:
  - i) Onscreen help that explains how completing a maze subsequently generates a larger maze for the player to solve.
  - ii) Image of a larger maze.
- j) The *end of level menu* should contain:
  - i) Text congratulating the user on their completion of the maze.
  - ii) Text that contains the time taken for the user to complete the maze.
  - iii) Text that contains the number of steps taken for the user to complete the maze.
  - iv) The next level button, which allows the user to proceed to a newly generated maze.
  - v) The quit to main menu button, a button that allows the user to quit to the main menu.
- k) The *pause menu* should contain:
  - i) The quit to main menu button, a button that allows the user to quit to the main menu.
  - ii) Small amount of text expressing that pressing the escape key will return the user to the maze.
- l) When the game is executing and a maze being displayed:
  - i) All of the game elements (such as tiles, the maze, player, exit tile, etc.) will be drawn to the screen with a HTML 5 canvas element.
  - ii) The canvas should be encapsulated within a 512 by 512 pixel bordered container.
  - iii) A grid of tiles with up to four walls on each side that represents the maze must be drawn to the canvas, along with a player and exit tile.
  - iv) The tiles in the game should be 32 by 32 pixels.
  - v) The player tile should be 32 by 32 pixels.
  - vi) The exit tile should be 32 by 32 pixels.
  - vii) The walls of the maze should be 32 by 2 pixels.
  - viii) The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of light grey.
  - ix) Saturation and lightness of player tile should remain constant at values of 80% and 40% respectively.
  - x) The colour of the exit zone should be orange with hex value #FF8000.
  - xi) If the maze fits in the canvas (512 by 512 pixels) then the maze should be centred in the canvas and the camera disabled.
  - xii) If the maze doesn't fit in the dimensions of the canvas (512 by 512 pixels) then a camera should be enabled that has the player positioned at the centre and follows the centre of the player, after panning has occurred.
  - xiii) Test that the colour of the player fades correctly over time. The player should fade from green to purple and then back again in a recurring cycle.
  - xiv) The player's trail must be displayed. Each tile visited by the player must be coloured with a lighter shade of the colour of the player.

**Signature of client**

*The signature below certifies that the client has read through the final design objectives and has agreed to each of the points. Once each objective has been tested, it can be proved then that the program functions as requested by the client.*



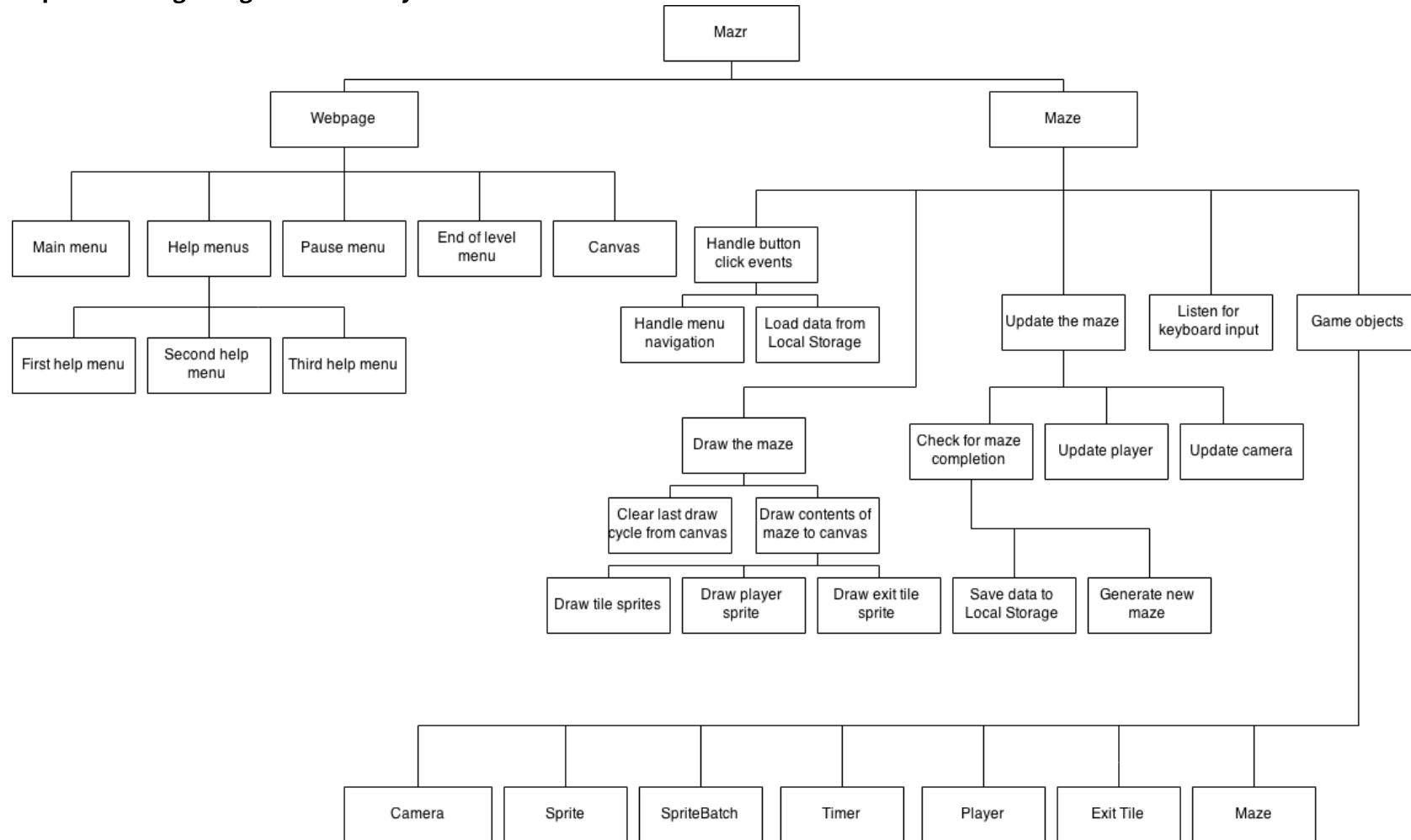
## Design documents

### Top down design for new system

Top down design is a paradigm in which a large system is subdivided repeatedly into small manageable modules that can be developed in isolation and then combined at a later stage. This allows for a complex piece of software to be split into much simpler programming tasks that can be developed and tested without interference or dependencies on other parts of the system.

The programming portion of Mazr will be written in JavaScript and embedded in a HTML webpage. The object-oriented features of JavaScript will be used in order to represent individual entities within the context of the program. For example, the Maze will be given its own object that encapsulates the state and behaviour of a maze entity in the game. Object Oriented Programming also involves modularisation – it allows objects to be developed independently and then combined in order to maintain modularity in the program.

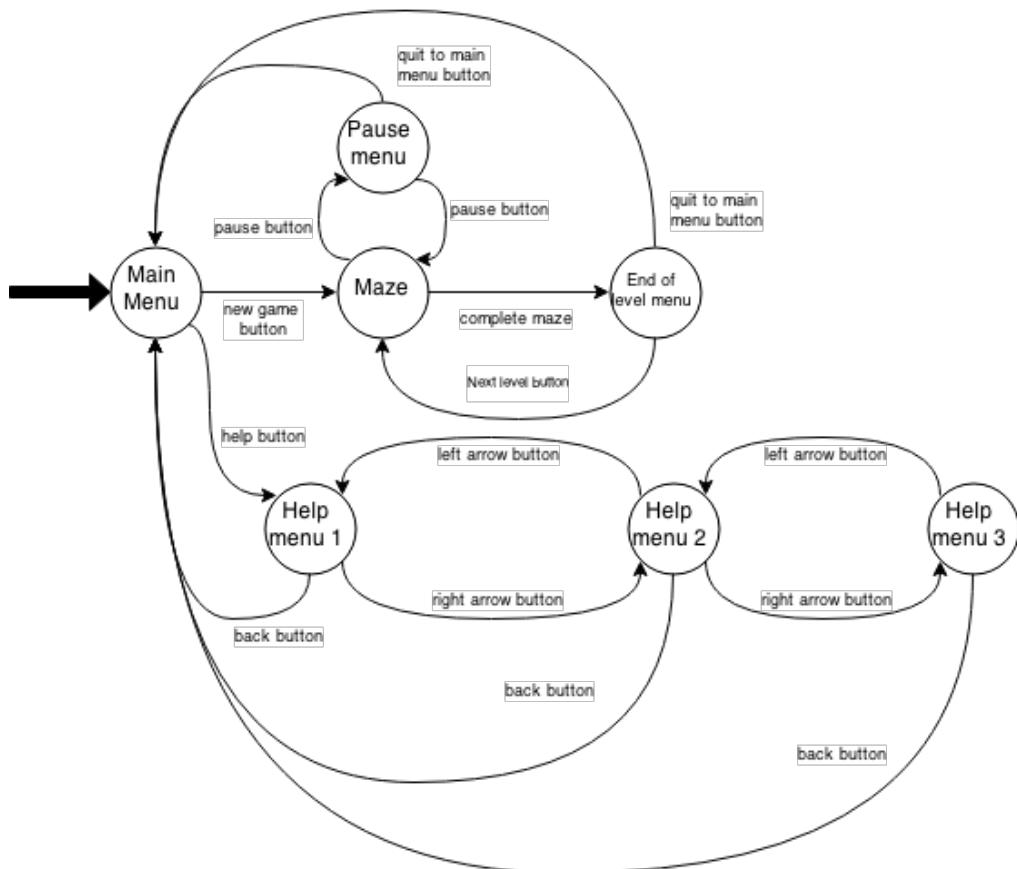
Below is a top down diagram that shows all of the different components that will be combined to develop the new system, Mazr.

**Top down design diagram for new system**

### Design of menu navigation

The design specification contains numerous objectives that mention the inclusion of menus in the game. These menus will all be contained within a single webpage and rendered in HTML. Only one menu will be visible at the time and no menus will be visible when a maze is being displayed. Therefore it needs to be defined as to how users can navigate between these menus and the actions that trigger this navigation.

Below is a finite state machine ([http://en.wikipedia.org/wiki/Finite-state\\_machine](http://en.wikipedia.org/wiki/Finite-state_machine)) that describes how the game is composed of the different menus.



A finite state machine is being used to model the display and navigation of menus because each state is discrete and can be used to model the currently displayed menu. Each state (a circle on the diagram) represents either a menu or in the case of 'Maze' the displaying of a maze. The arrows represent state transitions in the forms of various actions within the game that trigger navigation between menus. The Main Menu represents the initial state, which loads when the user visits the webpage and is represented by thick black arrow (the entry into the state machine). This information can be used to build a model of how the menus will exist in the game and how the user can navigate between menus to perform various functions.

This diagram is useful in defining the contents of the menus and behaviour of navigation. For example, it is trivial to note from the diagram that to navigate from the Main Menu to Help Menu 2 the user must press the help button to navigate to Help Menu 1 and then press the right arrow button to navigate to Help Menu 2.

### Use of libraries

Libraries are modules of code that have already been written and can be linked to a program that is being written in order to leverage the function of the libraries. They often contain routines common to programmers (such as searching or sorting) and are already pretested by the third party that developed them so are generally error free. This project will require the use of libraries in order to avoid “reinventing the wheel” and spending a long amount of time developing code that is available for free on the Internet. This will speed up testing and development time.

jQuery (<http://jquery.com>) is a JavaScript library that is very popular. It is a framework that is used to traverse and manipulate the DOM (document object model) of a HTML document. The DOM is a tree that represents all the elements present on a HTML page, which allows it to be changed and manipulated during the runtime of a JavaScript script. Although JavaScript can natively access the DOM, this can be tedious and take a long time and jQuery contains functions attached to the jQuery object that makes this much faster. Therefore this library will be used in the software development stage in order to handle the button click events that will provide navigation functionality between menus and any styling changes that must occur during the game’s runtime.

Reset.css (<http://meyerweb.com/eric/tools/css/reset/>) is a small CSS stylesheet file that is used in order to normalise the default appearance and styles of a webpage between browsers before any styling is implemented. This reduces any differences in the style of a webpage between various browsers and so will provide a more cohesive viewing experience on different platforms. This will be linked to the HTML in software development.

### Structuring the menus

The menus of the game will all be contained and displayed within a single HTML document that can be fetched with a web browser. This may seem counterintuitive as it would make sense for each menu to be a separate HTML document and the buttons that are used to navigate between the menus be links. However, this would require more HTTP requests on the part of the user in downloading the webpages from a server and would also require more files be created. Therefore, a different approach will be taken.

Each menu will be contained with a separate HTML div element. Divs are containers that will encapsulate the functionality of each menu. Each will be assigned a class tag so they can be styled and another class so they can be referenced. jQuery (see use of libraries above) will be used to reference these menus in JavaScript and allow them to be hidden and made visible when different events occur. For example, when the help button on the main menu is clicked, jQuery will hide the main menu and display the first help menu (as per the functionality stated in the finite state machine above).

### Storage Structure

As Mazr will be a web game contained within a HTML document, it will be accessed via web browser. Therefore, there will be no explicit file structures or saving data to the file system, as would be required with a traditional desktop-based application. Local Storage will be used as a substitute for files. HTML 5 Local Storage API (application programming interface) will be used in order to save game progress to a user’s computer locally. This eliminates the need for any backend or server system to save game data about different user. In this system, each user has data saved to their computer’s secondary storage. This means that each user has one computer on which their data is saved.

Local Storage uses a table of key value pairs to store data. A key object references each value saved to Local Storage. The key acts as an identifier to access the correct value data. Therefore, data about the game can be entered into Local Storage and will persist, even after the Mazr webpage is closed or refreshed. Below is a table that contains the keys that will be used in this program and a description of what the value will represent.

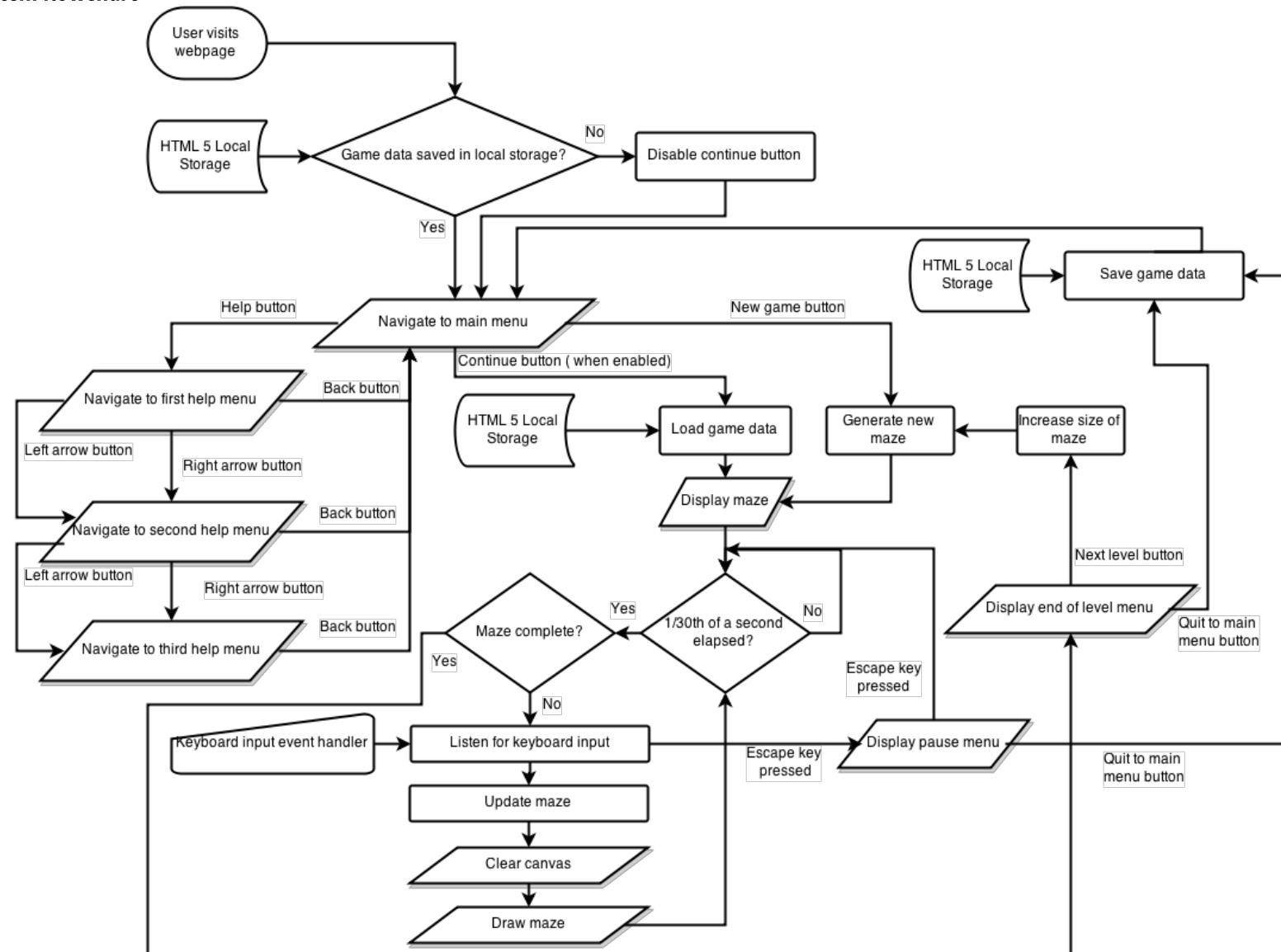
Key	Description of value
“level”	This contains the current level that the player is on. Allows the game to be resumed from this level.
“x”	The number of tiles in the maze in the x-axis when the game is saved.
“y”	The number of tiles in the maze in the y-axis when the game is saved.

### System Flowchart

The following page contains a diagram that represents the flowchart for the main system. The main components of the game and they work together to form a completed system is summarised in this diagram. The diagram uses a level of abstraction necessary to prevent over complication at this stage.

Note that the chart often uses labelled connectors to denote the path the system takes when a certain action is executed, such as the press of a certain button. This is used to save on space and to be more efficient instead of including a decision block at each stage in order to wait for the pressing of a button. It also fits naturally with how the system may use an event based triggering of control based on input.

The chart also has no termination point. This is because the game is only terminated when the browser window is closed and this is a functionality that is not within the boundaries of the game’s logic and so has no correct place on the following flowchart.

**System flowchart**

## Data structure design

Overleaf is a list of the data structures that will be used in Mazr. Each data structure is organised into its titled own table that contains information about its state and behaviour. State and behaviour represents data that an object in a program can maintain, for example in the form of attributes and methods. Below is an explanation of the notation used in the tables.

**Identifier** – This represents the unique identifier that will represent the individual state and behaviour that are used in the game. For example, the *Camera* data structure has an *attribute* (see below) with identifier *x*.

**Type** – This is the type of state or behaviour that will be implemented by the data structure. Do not confused with the data type (see further down). These include:

- **Attribute** – These are variables that are exposed to the rest of the program and are accessible outside of the data structure in which they are contained. Can be referenced outside of the object by the dot (.) operator.
- **Local Variable** – These are variables stored by the data structure but are not exposed to the rest of the system. They can only be accessed within the data structure.
- **Method** – These are functions that belong to the data structure. They can be called on the data structure. Note that these are included as part of the data structure as function objects because in JavaScript functions are actually objects.
- **Function** – These are methods that are not specific to a single data structure.
- **Getter Method** - These are methods (see above) that have a special purpose to appear like accessing a variable to the rest of the system. This way, they make a function call seem like a reference to a variable, which is a powerful way of expressing functionality.
- **Setter Method** – These are methods (see above) that have a special purpose to appear like assigning a value to a variable to the rest of the system. This way, they make a function call seem like the assignment of a variable.
- **Constant** – This is a local variable (see above) whose value cannot be changed after it has been assigned on its instantiation.

**Data Type** – This is the data type of the types above. For example, the *x* attribute of *Camera* has a Number data type. Note that these data types are based on those available in JavaScript. They could be Objects, or even Objects with defined data structures shown in this section (as will occur numerously in this project). For example, functions and methods don't have data types and will be labelled N/A. However, if they return a value then the data type of the return value will be included (along with a reminder this is a return type).

**Description** – This is a description of the state or behaviour and which it does or store as part of the data structure.

**Default Value** – This is the default value the state or behaviour holds. For functions and methods, this is N/A.

**Validation** – This is the validation rules that are applied to the state or behaviour to ensure it obeys certain regulations in the system.

**Data Structure Tables**

<b>Camera</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
x	Attribute	Number	The x coordinate of the camera	0	Must be a number object. Must not be null or undefined.
y	Attribute	Number	The y coordinate of the camera	0	Must be a number object. Must not be null or undefined.
width	Attribute	Number	The width of the camera's viewport	512	Must be a number object. Must not be null or undefined.
height	Attribute	Number	The height of the camera's viewport	512	Must be a number object. Must not be null or undefined.

<b>Sprite</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
x	Attribute	Integer	The x coordinate of the sprite	0	Must be a number object. Must not be null or undefined.
y	Attribute	Integer	The y coordinate of the sprite	0	Must be a number object. Must not be null or undefined.
colour	Attribute	String	A string of hexadecimal digits that determines the colour of the sprite	"#000000"	Must be of type string. Must not be null or undefined.
walls	Attribute	Array	An array of four Boolean values. Each element in the array stores whether one of the four walls of the tile is present. Ordered north wall, east wall, south wall, and	[true, true, true, true]	Must be an array. Each element is of type Boolean. There must be four elements in the array.

			west wall.		
--	--	--	------------	--	--

<b>SpriteBatch</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
batch	Attribute	Array	Array of Sprite objects that are to be drawn to the canvas.	Empty Array	N/A
canvasContext	Attribute	Object	Reference to the canvas drawing context onto which the batch is drawn	Undefined	Must not be undefined or nil
camera	Attribute	Camera	Reference to camera	Undefined	N/A
draw()	Method	Undefined (type of return value)	Method handles the drawing of sprites in batch	N/A	N/A

<b>SpriteBatch</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
lastTime	Local Variable	Number	The last time recorded by the timer.	Current clock tick.	N/A
elapsedTime	Local Variable	Number	The elapsed time since last check.	0	N/A
start()	Method	Undefined (type of return value)	Starts the timer. This is not explicitly required as instantiating a Timer object starts the timer automatically.	N/A	N/A
run()	Method	Boolean	Returns true after a parameter of time has elapsed	N/A	N/A

<b>ExitTile</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
x	Getter method	Integer	Reference to the x attribute of sprite attribute	this.sprite.x	N/A
y	Getter method	Integer	Reference to the y attribute of	this.sprite.y	N/A

			sprite attribute		
x	Setter method	Integer	Reference to set value of the x attribute of sprite attribute	this.sprite.x	N/A
y	Setter method	Integer	Reference to set the value of y attribute of sprite attribute	this.sprite.y	N/A
sprite	Attribute	Sprite	The sprite that is drawn that represents the exit tile	N/A	Must be a Sprite object.

<b>Player</b>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
x	Getter method	Integer	Reference to the x attribute of sprite attribute	this.sprite.x	N/A
y	Getter method	Integer	Reference to the y attribute of sprite attribute	this.sprite.y	N/A
x	Setter method	Integer	Reference to set value of the x attribute of sprite attribute.	this.sprite.x	N/A
y	Setter method	Integer	Reference to set the value of y attribute of sprite attribute	this.sprite.y	N/A
newPositionY	Attribute	Integer	When the player's position is moved, this y value is updated. The player is then animated until this y value is reached.	x property	N/A
newPositionX	Attribute	Integer	When the player's position is moved, this x value is updated. The player is then animated until this x value is reached.	y property	N/A
sprite	Attribute	Sprite	The sprite that is drawn that represents the player.	N/A	Must be a Sprite object.
disabled	Attribute	Boolean	Stores whether the player's movement is disabled	False	N/A
animatePosition()	Method	Undefined (type of return value)	Updates the x and y properties of the player in small	N/A	N/A

			increments to match the newPositionX and newPositionY attributes.		
hueOne	Local variable	Integer	The minimum hue value the player can fade to	145	N/A
hueTwo	Local variable	Integer	The maximum hue value the player can fade to	300	N/A
currentHue	Local variable	Integer	The current hue of the player	hueOne	N/A
dHue	Local variable	Decimal	The change in hue in one step	(hueTwo – hueOne)/200	N/A
currentSaturation	Local variable	Integer	The current saturation of the player	80	N/A
currentLightness	Local variable	Integer	The current lightness of the player	40	N/A
fadeDirection	Local variable	Boolean	Whether the player is fading with increasing or decreasing hue values.	True	N/A
fadingTimer	Local variable	Timer	A timer which causes the hue to fade in equal time increments	New Timer Object	N/A
fadeColour()	Method	String	Function will increment the current hue by dHue depending on direction of change in hue and return the colour as an HSL string.	N/A	N/A
trailColour	Getter Method	String	Returns a HSL string representation of the trail colour of the player.	N/A	N/A

<b>Maze</b>						
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>	
complete	Property	Boolean	Stores whether the maze is complete	False	N/A	
maxX	Property	Integer	Maximum number of cells in x dimension	4	N/A	

maxY	Property	Integer	Maximum number of cells in y dimension	4	N/A
tileSize	Property	Integer	The size of one side of a square tile in the maze.	32	N/A
cells	Property	Array	Array of cells that make up the maze	New Array	N/A
camera	Property	Camera	Reference to camera which follows the player	Camera parameter	N/A
playerX	Local Variable	Integer	Stores the x coordinate of the player's position. Generated randomly.		N/A
playerY	Local Variable	Integer	Stores the y coordinate of the player's position. Generated randomly.		N/A
player	Property	Player	The player in the maze instance.		N/A
spriteBatch	Property	SpriteBatch	The sprite batch that controls the drawing of all sprites that make up the maze.		N/A
animationTimer	Property	Timer	Ensures all animations run in timed intervals.	New Timer	N/A
movementTimer	Property	Timer	Ensures the player movement occurs in timed intervals.	New Timer	N/A
panningTimer	Property	Timer	Ensures the camera pans in timed intervals.	New Timer	N/A
completionTimer	Property	Timer	Keeps a counter of the timer taken to the complete the maze.	New Timer	N/A
completedSeconds	Property	Integer	Stores the number of seconds taken to complete the maze	0	N/A
panning	Local Variable	Boolean	Stores whether the camera is currently panning independent of player movement	True	N/A
fixedMap	Local Variable	Boolean	Stores whether the level fits within the camera window and therefore disables the camera	False	N/A
initialise	Method	Undefined	Method populates data of new maze. All initialisation is encapsulated within this method. This is so maze can be reinitialized when new mazes are loaded	N/A	N/A
newMaze()	Method	Array	Generates a new maze using randomised depth first search	N/A	N/A
draw()	Method	Undefined (type of	Handles drawing of all sprites in maze	N/A	N/A

		return value)			
update()	Method	Undefined (type of return value)	Updates all of the game logic in the maze.	N/A	N/A
updatePlayerPosition()	Method	Undefined (type of return value)	This method waits to be called by the input event handlers and handles updating the position of the player within the maze.	N/A	N/A
colourTrail()	Method	Undefined (type of return value)	Updates the colour of the tile the player resides upon as the trail colour.	N/A	N/A
randomFloor()	Method	String (type of return value)	Returns a CSS colour string for a random shade of grey for use in the random floor.	N/A	N/A

<b>GAME (object literal)</b>					
<i>This is an object literal that will control the drawing, updating, saving and loading of a game in the global scope</i>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
save	Method	Undefined (type of return value)	Saves the current game level and dimensions in Local Storage.	N/A	N/A
load	Method	Undefined (type of return value)	Loads the current game level, width, height saved properties from Local Storage and saves them in level, mazeWidth and mazeHeight respectively.	N/A	N/A
update	Method	Undefined (type of return value)	One cycle of the update loop. Executed 30 times per second. Updates maze, checks for completion and handles generation of new mazes if necessary.	N/A	N/A
draw	Method	Undefined (type of return value)	One cycle of the draw loop. Executed 30 times per second. Clears the result of last draw cycle then calls method that draws updated maze to canvas.	N/A	N/A

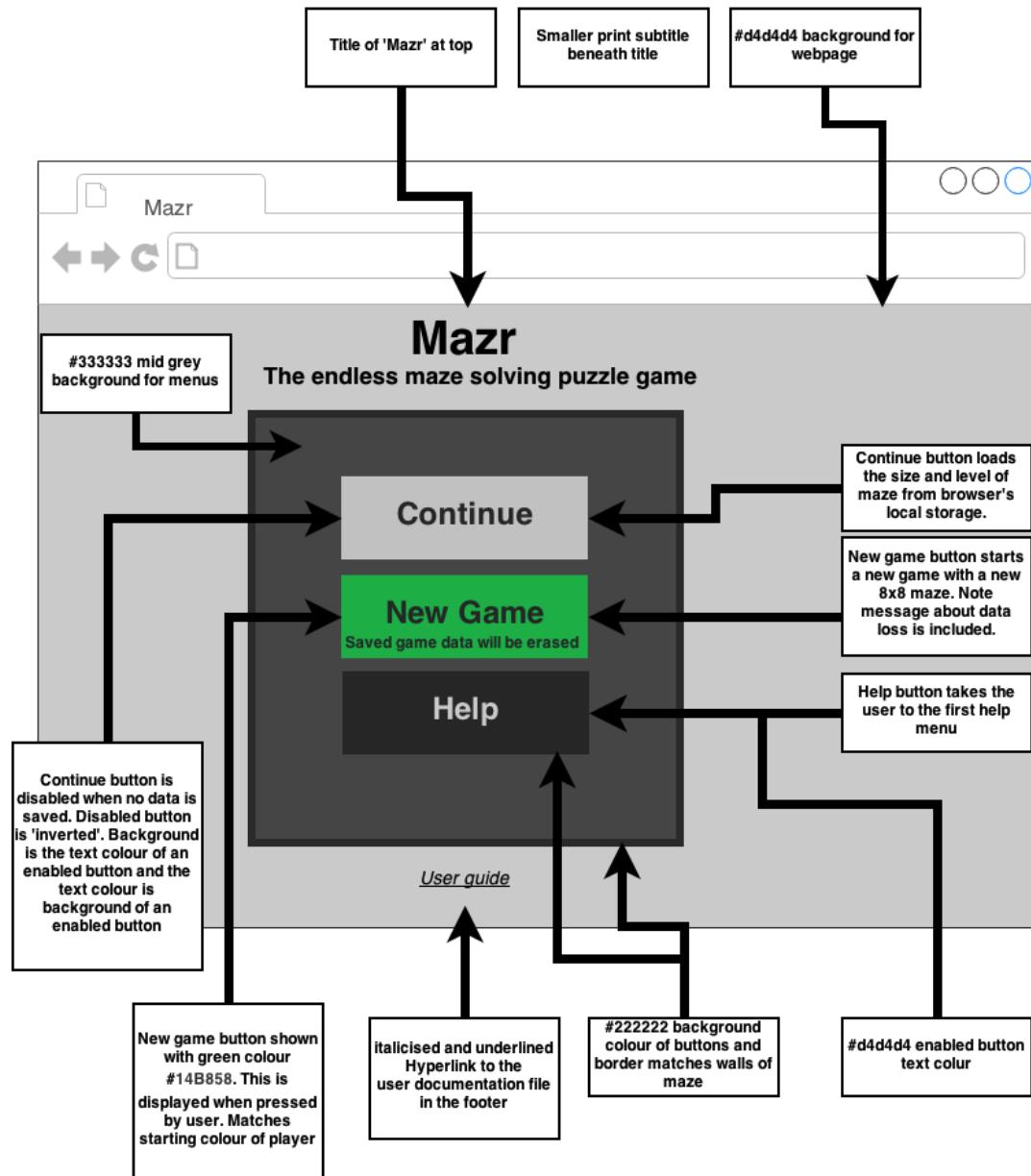
<b>Global Scope</b>					
<i>This is the global scope of the game. It will be contained within one file. It is not its own data structure but rather a collection of instances of objects etc.</i>					
<b>Identifier</b>	<b>Type</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Validation</b>
WIDTH	Constant	Integer	Width of the canvas	512	N/A
HEIGHT	Constant	Integer	Height of the canvas	512	N/A
FPS	Constant	Integer	Fixed frames per second	30	N/A
TILE_SIZE	Constant	Integer	The size of one side of the square tiles	32	N/A
STATE	Constant	Object (literal)	An object that acts as an enumeration of states that can be used to control whether player is on a menu or solving a maze.	{MENU: 0, MAZE: 1}	N/A
BACKGROUND_COLOUR	Constant	String	The background colour of the maze as a hex string	"#333333"	N/A
WALL_COLOUR	Constant	String	The colour of the walls in the maze as a hex string	"#222222"	N/A
EXIT_COLOUR	Constant	String	The colour of the exit of the game as a hex string	"#ff8000"	N/A
canvas	Local variable	Object	Reference to the canvas element	Undefined	N/A
canvasContext	Local variable	Object	Reference to the draw context of the canvas object	N/A	N/A
mazeWidth	Local variable	Integer	Width of the maze	4	N/A
mazeHeight	Local variable	Integer	Height of the maze	4	N/A
level	Local variable	Integer	The current level the player is on	1	N/A
maze	Local variable	Maze	The object that represents the maze in the game and retains associated state and functionality.	N/A	N/A
currentState	Local variable	Integer	The current state that the game is in. Designed to be used in conjunction with STATE object	STATE.MENU	N/A

			literal.		
pageStack	Local variable	Stack	Contains a stack of the menus the user has navigated to in the game.	N/A	N/A
showMenu	Function	Undefined (type of return value)	Displayed the menu that is passed with a string identifier in the parameter.	N/A	N/A
navigateTo	Function	Undefined (type of return value)	Navigates the user to the menu passed as a string identifier parameter.	N/A	N/A
goBack	Function	Undefined (type of return value)	Navigates back to the last menu in the stack of menus.	N/A	N/A

## Design of screen layouts

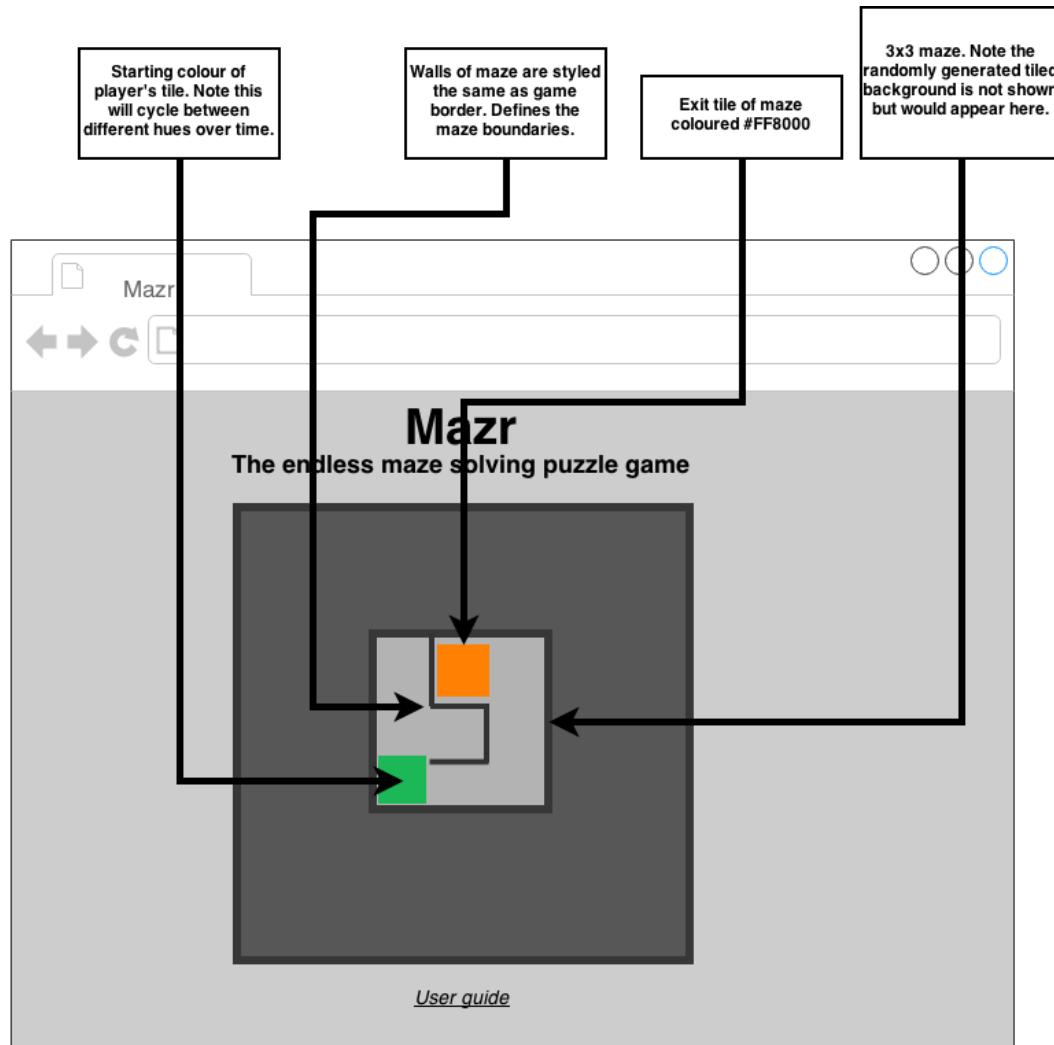
### Main menu design

This shows the mock-up for the design of the main menu. It shows the various elements of the menu and the different styles of the different states.



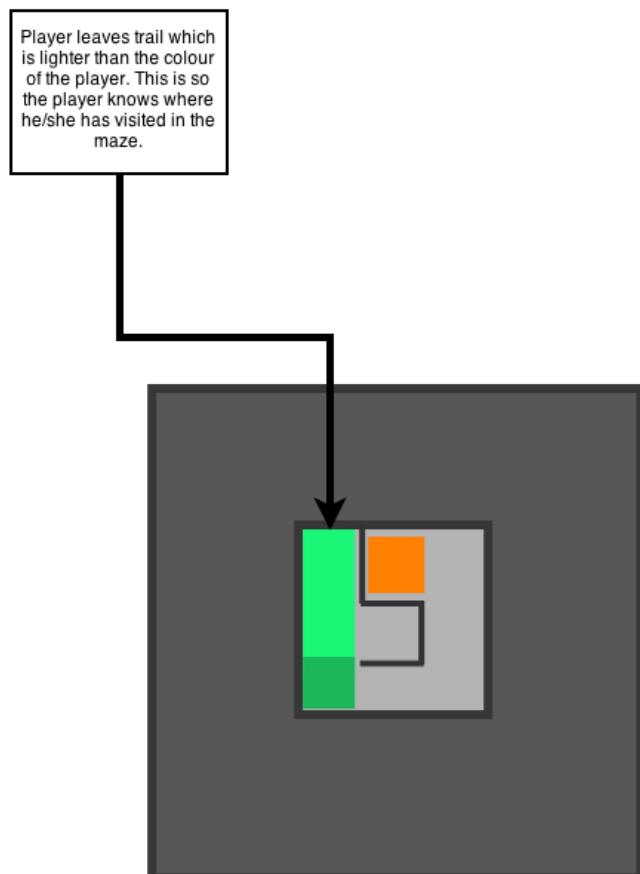
**View of maze**

This is a view of the maze when the user is attempting to solve it. Note that a pattern of grey tiles that was specified in the Design Specification in the maze is not included in this initial, simplified mock-up. It will however be integrated during Software Development in order to feature in the finished system.



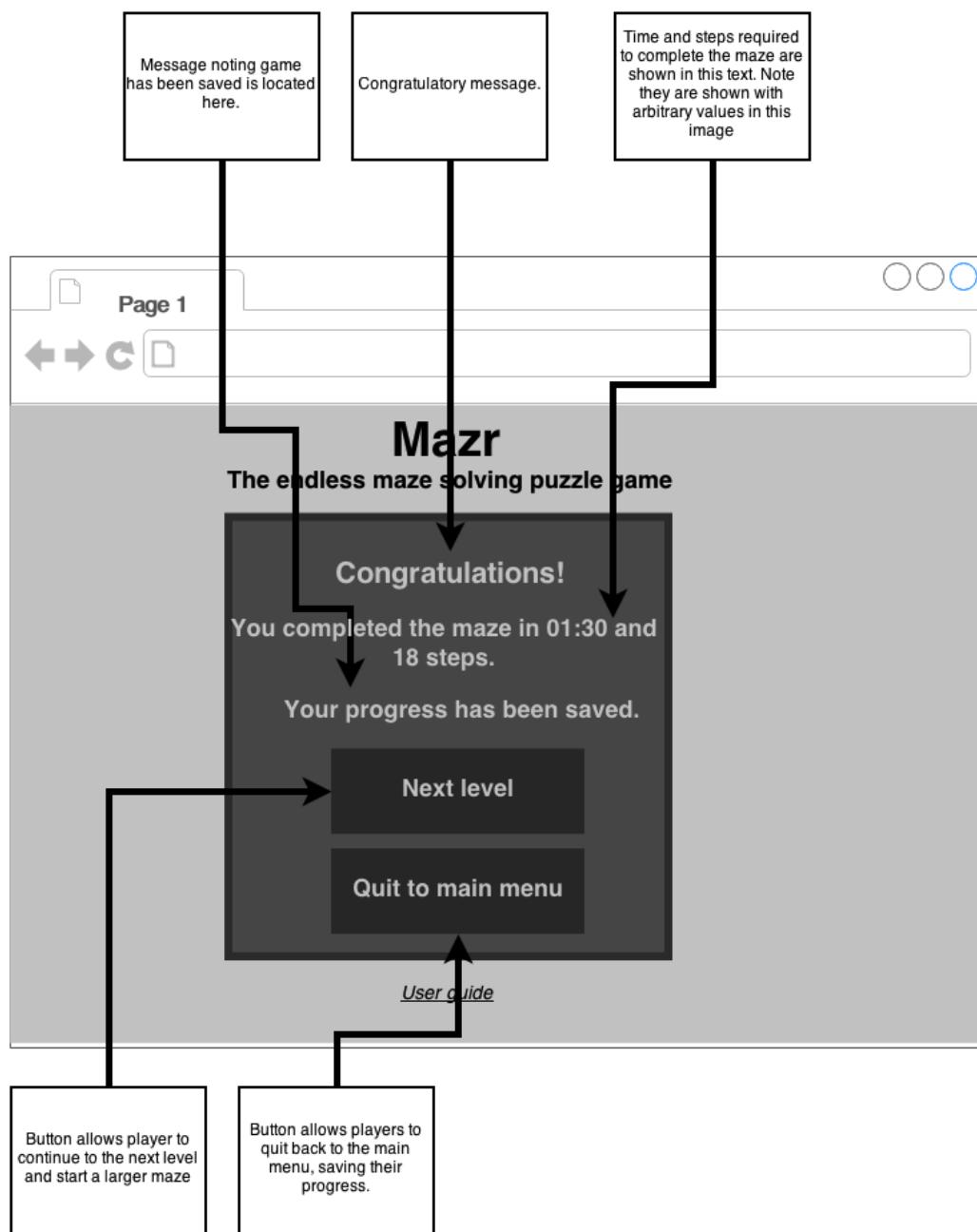
**View of player's trail in maze**

This image shows a simple view of how the trail will appear when the player moves. The trail allows the player, on larger mazes that may appear difficult, easily locate areas they have already solved to prevent large amounts of backtracking. Note that the colour of the trail is a lighter version of the colour of the player.



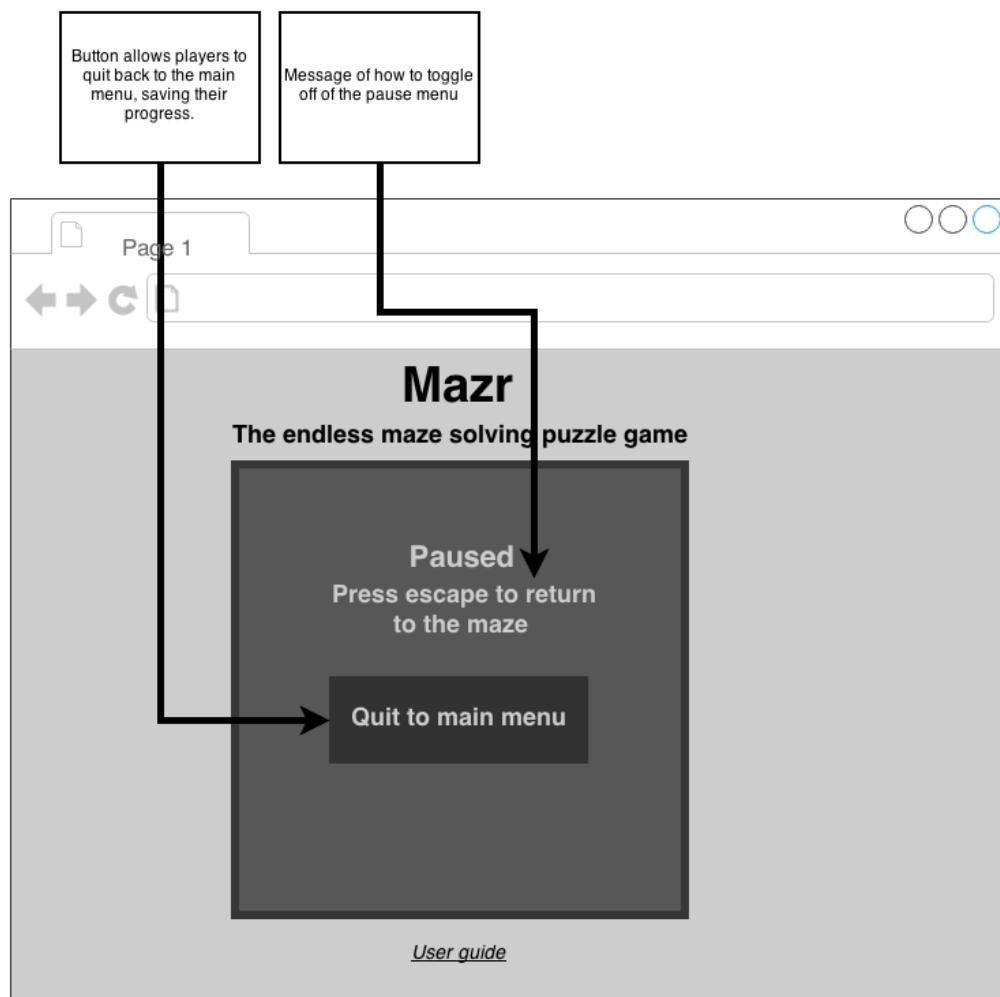
### End of level menu

The end of level menu is displayed when a maze is completed. It contains some information about the maze just completed and buttons to move to the next level or to return to the main menu.



**Pause menu**

The pause menu appears whenever the escape key is pressed on a maze. This pauses the timer and allows the player to take a break when solving the maze.



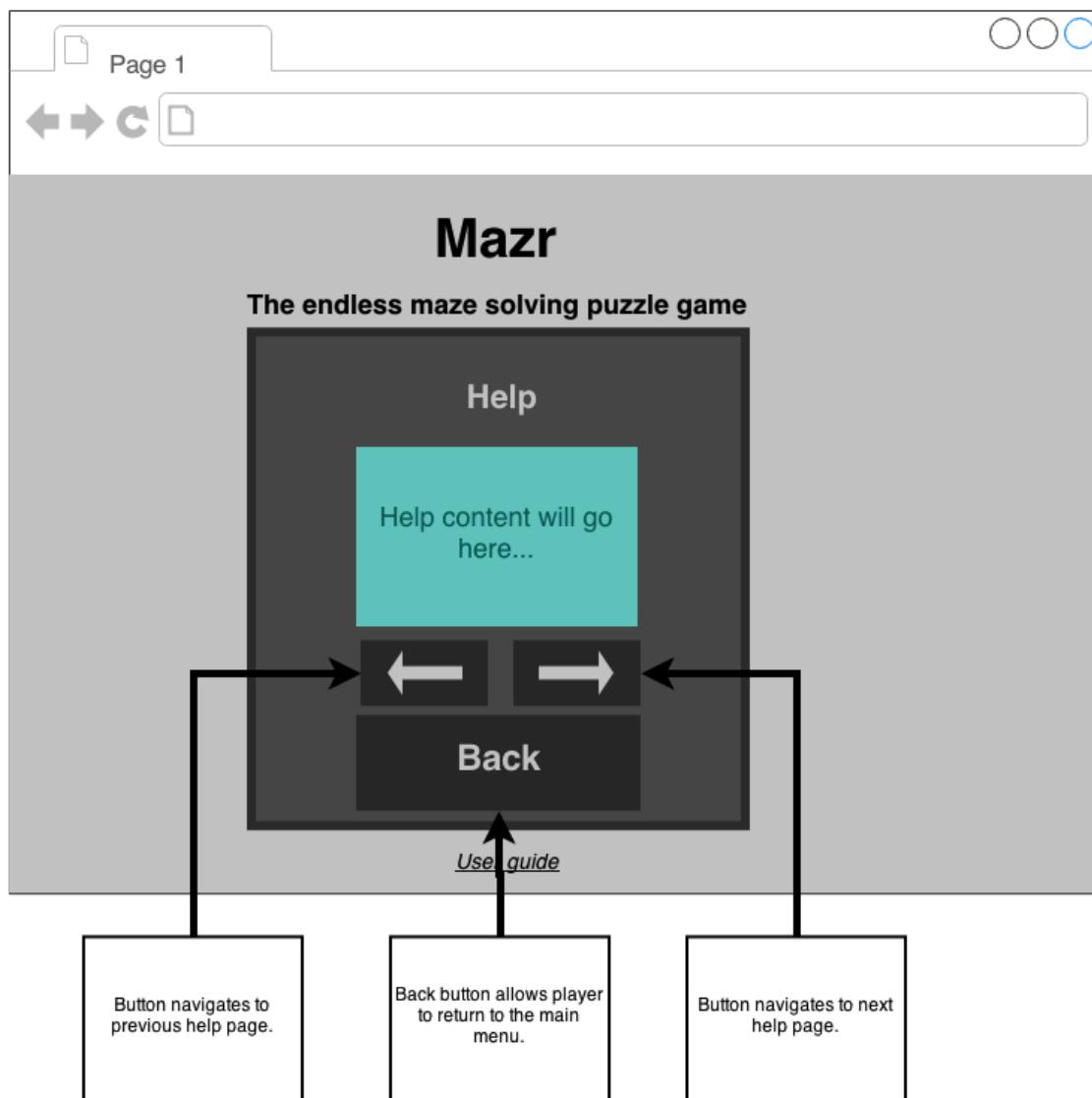
**JavaScript disabled error message**

This is an error message that will be displayed when JavaScript isn't displayed in the browser. The menus will all be hidden so that the user must enable the JavaScript to play the game.



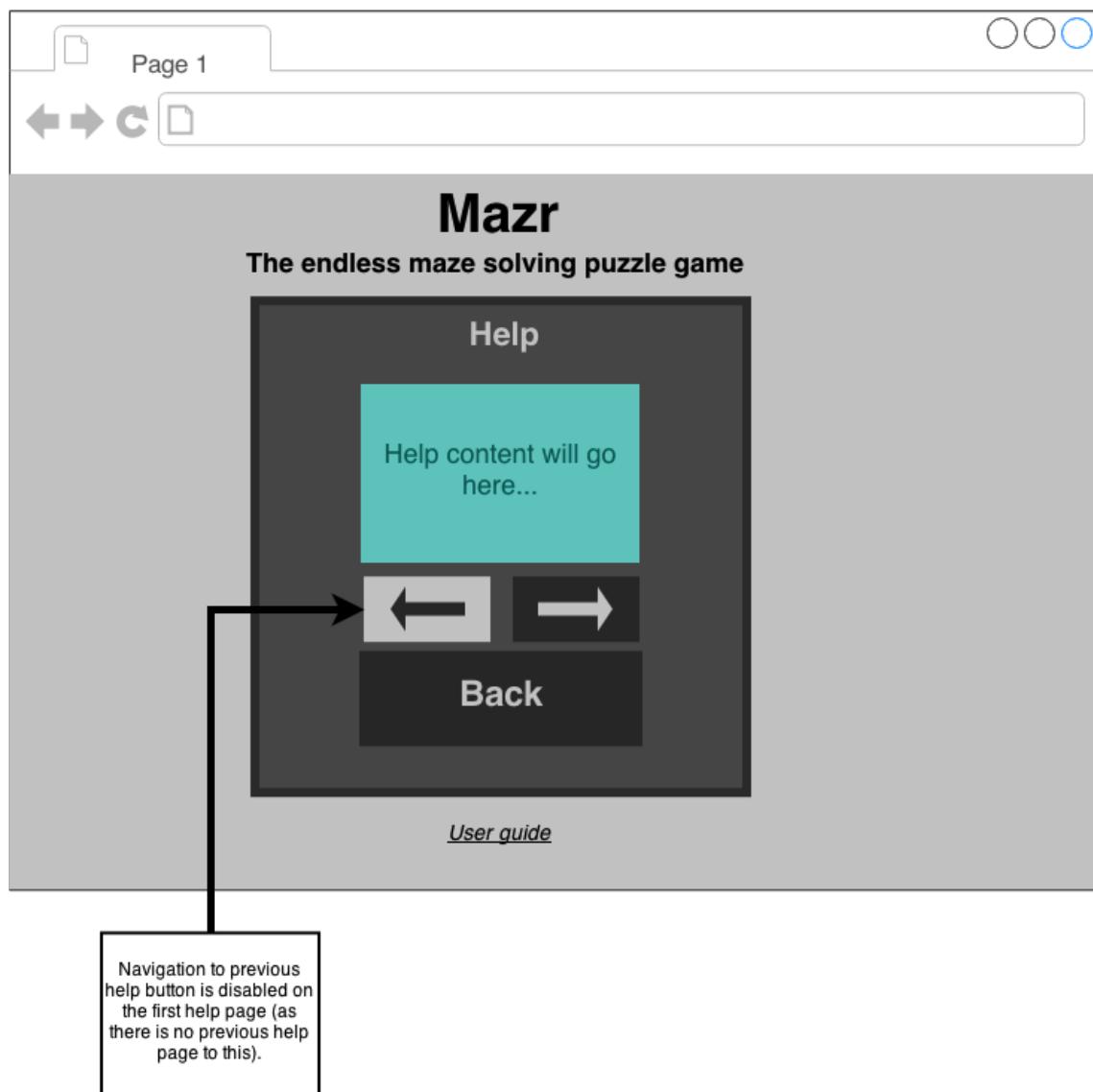
**Help menu one**

The first help menu is the one navigated to when the help button on the main menu is pressed. Arrow buttons can be used to navigate between the help menus. The help content is currently not present so a placeholder is included.



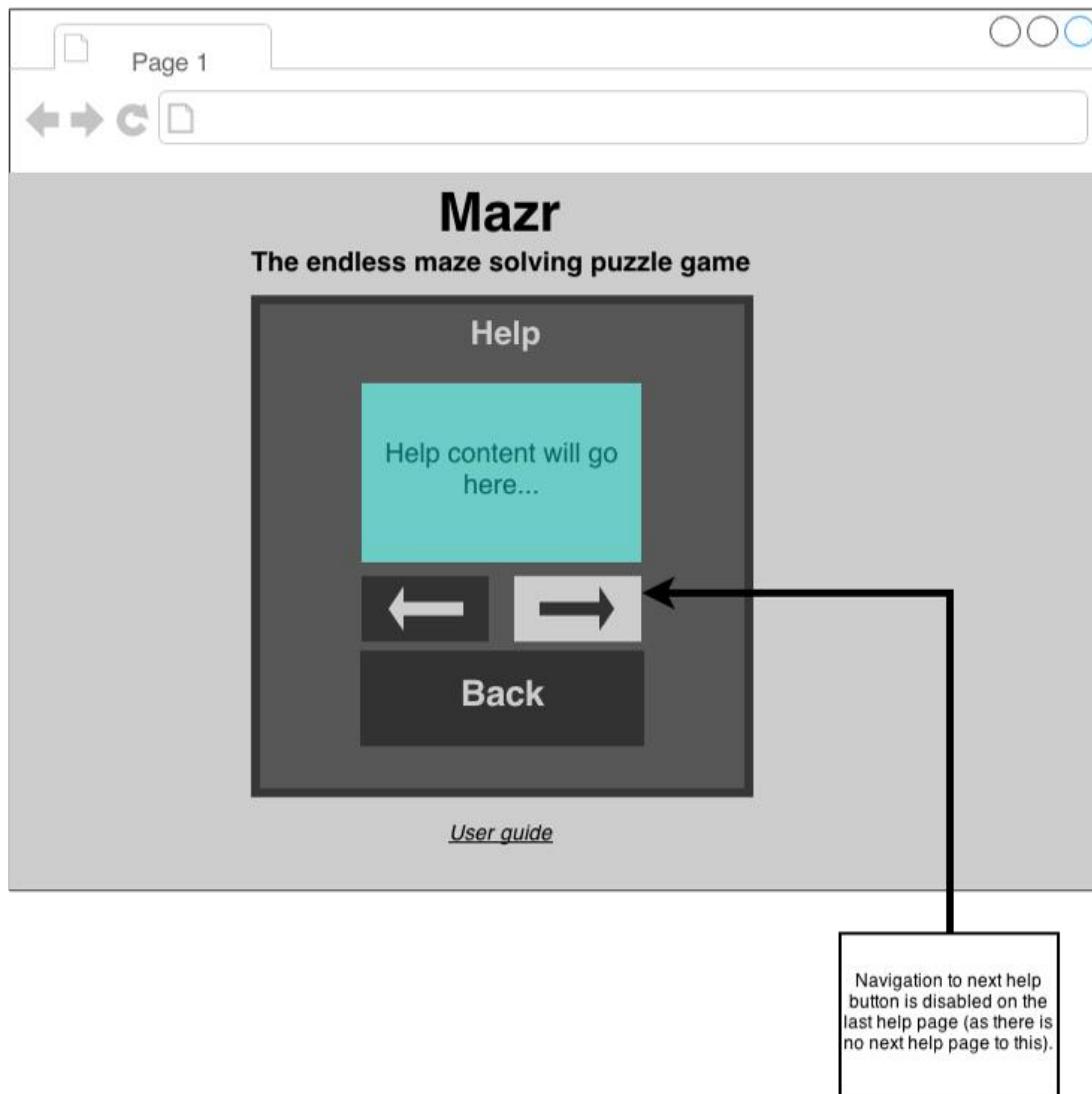
**Help menu two**

The second help menu is located after the first and presented when the right arrow button is pressed. The content of the help menu is shown with a placeholder.



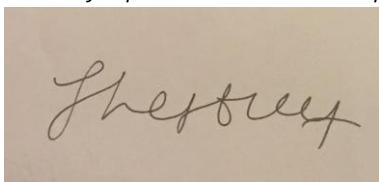
**Help menu three**

The third help menu is the last help menu and contains a placeholder where help content will be located in the future.



**Agreement to screen layouts with client**

*The signature below certifies that the client agrees that the screen layouts are satisfactory and will be faithfully reproduced in the developed system.*

**Contents of the help menus**

In the previous screen layouts, the contents of the onscreen help on the help menus were omitted and contained a placeholder. The content of the help menu will be summarised in the table below. Each help menu will contain a small amount of instructional text and an associated image.

Help Menu	Content of text	Description of the image that will be used
First help menu	You are the coloured square.  You are placed in a randomly generated maze. All mazes are solvable. Move onto the orange square to complete the maze.	Cropped screenshot of a first level 4 by 4 maze with the exit tile and player, no movement having been taken.
Second help menu	You can move around the maze with the arrow keys, providing no walls block your path. You will leave a coloured trail.	Cropped screenshot of a first level 4 by 4 maze with the exit tile and player, with the player having moved over to a tile adjacent to the exit tile and having left a trail.
Third help menu	When you complete a maze, a new, larger maze is generated.	Cropped screenshot of a second level 6 by 6 maze with the exit tile and player, with the player having moved around the maze and left a large amount of trail colour and now moved over to a tile adjacent to the exit tile.

## Styling the user interface with CSS

Now that the screen layouts have been designed it is possible to begin to develop an idea of the styling that will need to take place. Below is a table that contains all of the CSS information that will be placed into a stylesheet in order to achieve the screen layouts above.

Html Element	Selector	Property	Value
Html		box-sizing	border-box
		cursor	none
*, *:before, *:after		box-sizing	inherit
noscript		font-size	28px
		font-weight	normal
		style	italic
		color	red
		width	512px
		display	block
		margin	auto
body		background	#dddddd
		color	#222222
		font-family	Helvetica
		font-weight	bold
		text-align	center
		margin-left	auto
		margin-right	auto
hgroup h1		font-size	100px
hgroup p		font-size	16px
hgroup, footer		padding	12px 0px
h2		font-size	30px
p		padding	0px 50px
p	.under-text	font-size	12px
canvas		border	4px solid #222222
canvas	.page	padding	0
		margin	0
div	.page	background	#333333
		display	inline-flex
		border	4px solid #222222
		color	#dddddd
		width	520px
		height	520px
		margin	auto
button		.buttons	margin auto
		display	block
		margin	12px auto
		padding	8px 12px
		width	300px
		background	#222222
		border	0px
		padding	20px
		font-size	30px
		font-family	Helvetica
		font-weight	bold
	:active	color	#d4d4d4
	:active	outline	none

		box-shadow	inset 0px 2px 10px 2px rgba(0,0,0, 0.4);
		-webkit-box-shadow	inset 0px 2px 10px 2px rgba(0,0,0, 0.4);
		-moz-box-shadow	inset 0px 2px 10px 2px rgba(0,0,0, 0.4);
:focus	outline	none	
	color	#222222	
	background	hsl(145, 80%, 40%)	
:disabled	outline	none	
	background	#444444	
	color	#222222	
.btn-arrow	display	inline	
	width	150	
	border	5px solid #333333	
	margin	0	
a		color	#222222
		font-style	italic
		font-weight	lighter

# Algorithms

## Introduction

This section contains all of the algorithms that will be used as models of computation in order to program the new system.

The algorithms are written in *pseudo-code*, a way of describing computation to a human rather than a computer. They contain the structure of a program while the expressiveness of natural human language.

The algorithms operate on the Data Structures contained in the *Data Structures* section of *Nature of the solution*. Therefore, there are variable names that represent variables in the data structures section. The algorithms in this section represent all of the methods in that section.

Not all of the variables that are used in the algorithm will be contained in the data structures. This is because they are local in scope to the methods and functions defined in the data structures section, and so do not take up a place in the algorithm's respective data structure.

**Maze Generation Algorithm**

This algorithm shows how a maze will be generated with a randomized depth-first search algorithm. In essence, the program takes a random walk through a maze, knocking down walls of random neighbours and then backtracking when neighbours with knocked-down walls are found. This is done until every cell in the maze is visited. Thus, a perfect maze is generated as each cell can be reached from any other cell.

This algorithm will use data from the Maze data structure so refer to that data structure to find the meaning of certain variables.

```
FUNCTION newMaze
    tiles ← GRID Of maxX * maxY TILES WITH 4 INTACT WALLS
    tileStack ← EMPTY Last-In-First-Out STACK OF TILES
    totalTiles ← maxX * maxY
    visitedTiles ← 1
    currentTile ← SELECT RANDOM TILE FROM tiles

    WHILE visitedTiles < totalTiles
        neighbours ← ALL NEIGHBOURS OF CURRENT TILE IN tiles WITH
                      INTACT WALLS

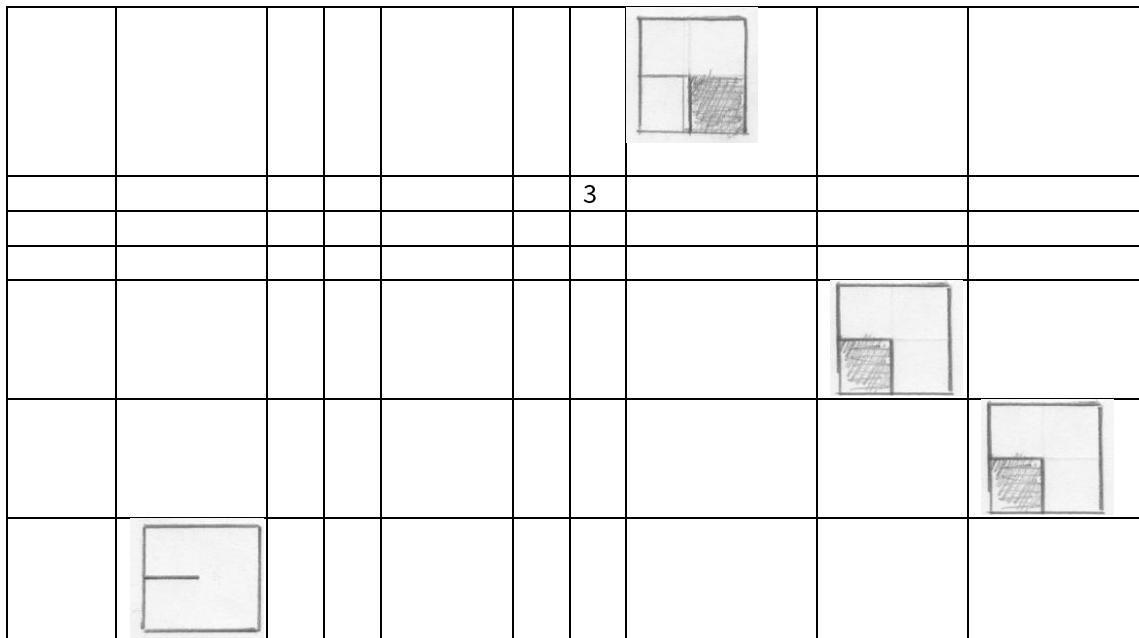
        IF LENGTH OF neighbours > 1 THEN
            selectedTile ← RANDOM TILE FROM neighbors

            KNOCK DOWN WALL BETWEEN currentTile AND selectedTile
            PUSH currentTile LOCATION ONTO tileStack
            currentTile ← selectedTile

            visitedTiles ← visitedTiles + 1
        ELSE THEN
            currentTill ← POP TILE OFF tileStack
        END IF
    END WHILE
END FUNCTION
```

**Maze Generation Algorithm Testing**

Line No	tiles	maxX	maxY	tileStack	totalTiles	visitedTiles	currentTile	neighbours	selectedTile
		2	2						
					4				
						1			
					[0,0]				
7.5					2				
					[0,0] [1,0]				



**Maze Initialisation Algorithm**

Below is the function that initializes the Maze object, which can be found in the data structures section of the Nature of The Solution – see there for more information on the data structure and how the algorithm manipulates it.

```
FUNCTION initialise
    cells ← GENERATE NEW RANDOMISED MAZE

    randomExitX ← PICK A RANDOM EXIT X-COORDINATE ONE THIRD OF THE
                  DISTANCE OF THE MAZE AWAY FROM THE PLAYER

    randomExitY ← PICK A RANDOM EXIT Y-COORDINATE ONE THIRD OF THE
                  DISTANCE OF THE MAZE AWAY FROM THE PLAYER

    CREATE A NEW OVERLAY ExitZone AT (randomExitX, randomExitY)

    PUSH NEW ExitZone TO overlays
    PUSH THE player TO overlays

    PUSH THE SPRITE OF EACH CELL TO spriteBatch
    PUSH THE SPRITE OF EACH OVERLAY IN overlays TO spriteBatch

    IF PIXEL WIDTH OF THE MAZE > WIDTH OF CAMERA THEN
        Panning ← FALSE
        fixedMap ← TRUE

        POINT THE CAMERA AT THE CENTRE OF THE MAZE
    ELSE
        POINT THE CAMERA AT CENTRE OF POINT(randomExitX, randomExitY)
    END IF
END FUNCTION
```

**Draw function**

This small function draws all the sprites belonging to a maze. This is a function of the Maze data structure.

```
FUNCTION draw
    DRAW EACH sprite IN spriteBatch
END FUNCTION
```

**Colour Trail of player function**

This small function colours the sprite of the tile the player is currently on with a light variant of the accent colour. This function belongs to the Maze data structure.

```
FUNCTION colourTrail
    colour OF sprite OF cells[x OF player][y OF player] ← CALL trailColour() ON player
END FUNCTION
```

**Update function of maze logic**

This function updates all the logic associated with the playing of the maze. This is a method of the Maze data structure.

```
FUNCTION update
    EVERY 1000ms DO
        completionSeconds ← completionSeconds + 1
    END

    Colour OF sprite OF player ← CALL fadeColour() ON player

    IF NOT panning THEN
        EVERY 10ms DO
            UPDATE THE POSITION OF THE PLAYER
        END

        IF NOT fixedMap THEN
            POINT THE CENTRE OF THE CAMERA ON THE PLAYER
        ELSE THEN
            EVERY 10ms DO
                IF THE CAMERA IS NOT CENTRED ON THE PLAYER THEN
                    MOVE THE CAMERA CLOSER TO THE PLAYER
                END IF

                IF THE CAMERA IS CENTRED ON THE PLAYER
                    panning ← FALSE
                END IF
            END
        END IF
    END IF

    complete ← TRUE, IF THE PLAYER HAS REACHED AN ExitZone

END FUNCTION
```

**Update the player's position function**

This function takes a string input of the direction in which to move the player. Some validation is performed to make sure the player can move. This is a method of the Maze data structure.

```
FUNCTION updatePlayerPosition(WITH PARAMETER direction OF TYPE STRING)
    IF THE player IS NOT ALREADY MOVING THEN
        COLOUR OF TILE UNDERNEATH THE player ← lightAccent OF graphics

        IF direction == "up" AND NO WALL IS ABOVE player THEN
            MOVE THE player UP ONE TILE

        ELSE IF direction == "down" AND NO WALL IS BELOW player THEN
            MOVE THE player DOWN ONE TILE

        ELSE IF direction == "left" AND NO WALL IS LEFT OF player THEN
            MOVE THE player LEFT ONE TILE

        ELSE IF direction == "right" AND NO WALL IS RIGHT OF player THEN
            MOVE THE player RIGHT ONE TILE

        ELSE THEN
            THROW AN ERROR WITH MESSAGE "Invalid Direction Parameter"
        END IF
    END IF
END FUNCTION
```

**Function That Waits for the Timer**

This function will be utilised by Timer objects in order to keep track of things that must change over certain periods of time. The wait function has to be *polled* by another function – effectively called repeatedly. It will return false on every poll until the time has elapsed and it will return true once, then return false for all polls until another elapse.

```
FUNCTION wait(WITH PARAMETER time OF TYPE NUMBER) RETURNS A BOOLEAN
    now ← GET CURRENT NUMBER OF CLOCK TICKS
    elapsedTime ← elapsedTime + (now - lastTime)
    lastTime ← now

    IF elapsedTime >= time THEN
        elapsedTime ← 0
        RETURN TRUE
    END IF
END FUNCTION
```

**Function that selects a random colour for floor tile**

The floor tiles are all assigned a random colour in order to make the game more visually appealing. All the tiles must be a shade of grey and this shade is selected by the function below. The function returns a string which represents the rgb format of the colour property that the tile will have.

The function belongs to a Maze object; see the data structures section for more information.

```
FUNCTION randomFloor
    g ← RANDOM NUMBER FROM 205 TO 255
    RETURN STRING "rgb(" + g + "," + g + "," + g + ")"
END FUNCTION
```

**Function animates the position of the player**

While the controls only allow the player to move from one tile to the next, assuming there are no walls to block this movement, the player's tile cannot simply be displayed in one and jump to the next. A smooth animation is necessary and this is achieved by keeping the player object keeping two variables for x and y – an absolute position in the maze representing the tile the player is on, and an animation position which represents where the player is.

This function effectively moves the player's animation position towards its absolute position until they are equal. This function belongs to the Player class.

```
FUNCTION updateAnimationPosition
    IF newPositionX > x THEN
        x ← x + TILE_SIZE / 4
    ELSE IF newPositionX < x THEN
        x ← x - TILE_SIZE / 4
    END IF

    IF newPositionY > y THEN
        y ← y + TILE_SIZE / 4
    ELSE IF newPositionY < y THEN
        y ← y - TILE_SIZE / 4
    END IF
END FUNCTION
```

**Draw function of SpriteBatch**

The SpriteBatch object will contain an array of Sprite objects that are to be drawn to the screen. They are to be drawn in the order they are placed in the array. So the 1<sup>st</sup> sprite needs to be drawn on top of the 0<sup>th</sup> sprite and so on. Note however that all of the walls are to be drawn on separately at the end as all other sprites reside within the walls.

```
METHOD draw()
    FOR i ← 0 UP TO (LENGTH OF batch - 1)
        IF batch[i].x - camera.x < camera.width AND
            batch[i].y - camera.y < camera.height THEN
            fillStyle OF canvasContext ← batch[i].colour
            canvasContext.fillRect(batch[i].x - camera.y, batch[i].y - camera.y, WIDTH, HEIGHT)

        END IF
    END FOR

    canvasContext.strokeStyle ← WALL_COLOUR
    canvasContext.lineWidth ← TILE_SIZE / 16

    canvasContext.beginPath()

    FOR i ← 0 UP TO (LENGTH OF batch - 1)
        IF batch[i].x - camera.x < camera.width AND
            batch[i].y - camera.y < camera.height THEN

            IF batch[i].walls[0] = TRUE THEN
                canvasContext.moveTo(batch[i].x - camera.x, batch[i].y - camera.y)
                canvasContext.lineTo(batch[i].x + TILE_SIZE, batch[i].y - camera.y)
            END IF

            IF batch[i].walls[1] = TRUE THEN
                canvasContext.moveTo(batch[i].x + TILE_SIZE, batch[i].y - camera.y)
                canvasContext.lineTo(batch[i].x + TILE_SIZE, batch[i].y - camera.y
                    + TILE_SIZE)
            END IF

        END IF
    END FOR
```

```
IF batch[i].walls[2] = TRUE THEN
    canvasContext.moveTo(batch[i].x - camera.x + TILE_SIZE, batch[i].y - camera.y
                        + TILE_SIZE)
    canvasContext.lineTo(batch[i].x - camera.x, batch[i].y - camera.y + TILE_SIZE)
END IF

IF batch[i].walls[3] = TRUE THEN
    canvasContext.moveTo(batch[i].x - camera.x, batch[i].y - camera.y + TILE_SIZE)
    canvasContext.lineTo(batch[i].x - camera.x, batch[i].y - camera.y)
END IF
END FOR
canvasContext.stroke()
END METHOD
```

**Control change of colour of player / trail**

In order for the game to be aesthetically pleasing, the colour of the player tile slowly fades through a variety of shades of colours. This change has to be simple and so a function will control the variation of this colour.

```
FUNCTION fadeColour
    IF 500ms HAS ELAPSED ON fadingTimer THEN
        IF currentHue >= hueTwo THEN
            upwards ← false
        ELSE IF currentHue <= hueOne THEN
            upwards ← true
        END IF

        IF upwards = true THEN
            currentHue ← currentHue + dHue
        ELSE THEN
            currentHue ← currentHue - dHue
        END IF
    END IF
END FUNCTION

FUNCTION trailColour
    return "hsl(" +
        currentHue +
        "," +
        (currentSaturation - 10) +
        "%," +
        (currentLightness+20) +
        "%)"
END FUNCTION
```

**Global draw function**

This is a small function that draws in the background and on top of which calls the draw function of the maze (This then draws the sprites held by the maze's spriteBatch).

This function is defined in the global scope and belongs to no classes or objects.

```
FUNCTION draw
    fillStyle OF canvasContext ← "#333333"
    FILL THE WHOLE OF canvasContext WITH RECTANGLE

    CALL draw ON maze
END FUNCTION
```

**Global Update Function**

This function handles the update cycle of the main game loop.

This function is defined in the global scope and belongs to no classes or objects.

```
FUNCTION update
    CALL update ON maze

    IF complete OF maze = true THEN
        SET ELEMENT OF ID 'time' EQUAL TO THE TIME TAKEN TO COMPLETE MAZE
        SET ELEMENT OF ID 'steps' EQUAL TO THE STEPS TAKEN TO COMPLETE MAZE
        maze ← NEW MAZE WITH DIMENSIONS mazeWidth, mazeHeight
        level ← level + 1

        INITIALIZE THE MAZE
        SAVE THE MAZE
    END IF
END FUNCTION
```

**Main Game Event**

The execution of the game begins here when the player visits the web page and the document loads (and considered ‘ready’). This is where all the main initialisation takes place and the keyboard event handler is registered. Then, the draw and update loops are executed 30 times a second. This function is in the global scope.

```
WHEN THE DOCUMENT IS READY
    COSTANT WIDTH ← 512
    CONSTANT HEIGHT ← 512
    CONSTANT FPS ← 30
    CONSTANT TILE_SIZE ← 32
    CONSTANT STATE = { MENU: 0, MAZE: 1 }
    CONSTANT BACKGROUND_COLOUR ← "#333333"
    CONSTANT WALL_COLOUR ← "#222222"
    CONSTANT EXIT_COLOUR ← "#ff8000"

    mazeWidth ← 4
    mazeHeight ← 4
    level ← 1

    GAME ← { EMPTY OBJECT LITERAL }

    canvas ← GET CANVAS ELEMENT FROM WEB PAGE
    canvasContext ← GET 2D CONTEXT FROM canvas

    currentState ← STATE.MENU

    maze ← NEW MAZE
    CALL initialise ON maze

    currentState OF stateMachine ← 1

    pageStack ← []

    EVENT LISTENER FOR KEYBOARD EVENTS DO
```

```
IF currentState OF StateMachine = 1 THEN
    IF THE UP ARROW IS PRESSED THEN
        MOVE THE PLAYER UP ONE TILE
    ELSE IF THE DOWN ARROW IS PRESSED THEN
        MOVE THE PLAYER DOWN ONE TILE
    ELSE IF THE RIGHT ARROW IS PRESSED THEN
        MOVE THE PLAYER RIGHT ONE TILE
    ELSE IF THE LEFT ARROW IS PRESSED THEN
        MOVE THE PLAYER ONE TILE LEFT
    END IF
ELSE IF currentState OF stateMachine = 2 THEN
    IF THE UP ARROW IS PRESSED THEN
        MOVE THE SELECTED ELEMENT UP BY ONE
    ELSE IF THE DOWN ARROW IS PRESSED THEN
        MOVE THE SELECTED ELEMENT DOWN BY ONE
    ELSE IF THE ENTER KEY IS PRESSED THEN
        ACTIVATE THE CURRENTLY SELECTED ELEMENT
    END
END IF
END

DO FPS TIMES PER 1000MS
    CALL update
    CALL draw
END
END
```

**Event handlers**

These are algorithms that display how the events that occur are handled when buttons are clicked in the game. Each is self explanatory and self contained.

```
WHEN THE CONTINUE BUTTON IS PRESSED THEN
    LOAD GAME DATA FROM LOCAL STORAGE
    maze ← NEW MAZE FROM DATA IN LOCAL STORAGE
```

```
    CALL initialise ON maze
    DISPLAY THE CANVAS
```

```
END
```

```
WHEN THE NEW GAME BUTTON IS PRESSED THEN
```

```
    level ←
    mazeWidth ← 4
    mazeHeight ← 4
```

```
    maze ← NEW MAZE WITH DIMENSIONS(mazeWidth,mazeHeight)
    CALL initialise ON maze
```

```
    SAVE GAME DATA TO LOCAL STORAGE
```

```
    DISPLAY THE CANVAS
```

```
END
```

```
WHEN THE HELP BUTTON IS PRESSED THEN
```

```
    DISPLAY THE FIRST HELP MENU
```

```
END
```

```
WHEN THE BACK BUTTON IS PRESSED THEN
    IF TOP OF pageStack IS A HELP PAGE THEN
        WHILE TOP OF pageStack IS A HELP PAGE THEN
            POP THE PAGE FROM THE STACK
        END IF

        NAVIGATE TO (POP page OFF pageStack)
    ELSE
        GO BACK A PAGE
    END IF
END

WHEN THE LEFT BUTTON IS PRESSED THEN
    IF THERE IS A PREVIOUS HELP MENU THEN
        GO BACK ONE HELP MENU
    END IF
END

WHEN THE RIGHT BUTTON IS PRESSED THEN
    IF THERE IS A FOLLOWING HELP MENU THEN
        GO TO NEXT HELP MENU
    END IF
END

WHEN THE NEXT LEVEL BUTTON IS PRESSED THEN
    GO BACK (TO CANVAS)
END
```

```
WHEN THE QUIT BUTTON IS PRESSED THEN
    WHILE LENGTH OF pageStack != 0 THEN
        POP PAGE OFF pageStack
    END WHILE

    SAVE GAME DATA TO LOCAL STORAGE

    NAVIGATE TO THE MAIN MENU
END
```

# Test Strategy

## Introduction

This section contains planning of how the system will be tested during the different stages of development. Different styles of testing will be used at the different stages as appropriate in order for the system to be considered functional.

Initial testing will take place during system software development and modifications will be made thereafter and is denoted as *Alpha testing*. Post development testing will then take place in order to evaluate that the system meets the criteria set out in the design objectives agreed to with the client in the *Nature of the solution*. Any test in the table that fails will cause the system to be modified and the test reevaluated for success. *End user testing* will involve the completion of a small questionnaire by a sample of end users that will be used to ensure Mazr is appropriate for its audience. Any modifications again will be made at this stage with respect to the end user's feedback. Finally, *Acceptance testing* will be completed. This is where the client evaluates the results of the alpha testing, the post development testing and the end user testing along with their own experience using the system. This will take place in the form of a short discussion based interview. The client may suggest possible modifications to the system that will be made. The client will finally sign off if she believes that the testing is completed to a satisfactory level to show that the system functions as was intended in the design objectives.

## Alpha testing

This is testing that is completed by myself while the software is being developed. As software development is an iterative process in which various modules of code are created independently of each other and then combined, the alpha testing will take a similar approach and each of the individual sections will be tested straight after they have been developed. This allows errors to quickly be rectified before small sections of code are integrated into the larger system as a whole and it becomes more difficult to identify the source of any bugs or errors.

Alpha testing will follow a simple plan but allow for flexibility. As not everything when developing software can be predicted, alpha testing will be taken with a more holistic approach and completed where it is seen fit during development. After each alpha test, should there be any errors or bugs, corrections to the software will be made appropriately and integrated into the system immediately.

The new system will be programmed using the OOP (object oriented programming) paradigm and as such the program will be split into classes from which objects can be instantiated. These are detailed in the Data Structures section of the *Nature of the solution*. Each of the elements of the program will be alpha tested individually. The table overleaf summarises the plan for the alpha testing. Note that as alpha testing will be completed where it seems appropriate in development this plan is merely guidelines on the alpha testing that will be completed and the actual alpha testing that will occur could deviate substantially from the plan as is seen fit. This table will be reproduced in the *Testing* section once the new system has been developed and the header reference column will be populated with header titles indicating presence of evidence of alpha testing in the *Software development* section of the report.

### Alpha test plan

Software area to be tested	Description of alpha test	Headers for alpha testing in Software Development
HTML webpage	Ensure that all the content is displayed on the webpage.	
CSS styling	Ensure the appropriate styling is applied to the webpage.	

Camera class	Run a test using the JavaScript console in a web browser in order to determine that an object of type Camera can be instantiated correctly.	
Sprite class	Run a test using the JavaScript console in a web browser in order to determine that an object of type Sprite can be instantiated correctly.	
SpriteBatch class	This will require the writing of additional test code that will be temporarily inserted into the JavaScript file and removed after this alpha test is complete. Run an initial test to ensure that a SpriteBatch object can draw rectangle sprites by drawing a grid of sprites. Run another test to ensure that a SpriteBatch object can draw walls by adding walls to the grid of sprites.	
Timer class	Write some test code in the Timer JavaScript file that will instantiate and run a one second timer that outputs the number of elapsed second to the JavaScript console of a web browser. Inspect whether these updates appear every second.	
Player class	Copy and modify the test code that was used for SpriteBatch in order to draw a player onto the grid of sprites. Test whether the player moves correctly and its colour fades correctly.	
ExitTile class	Run a test using the JavaScript console in a web browser in order to determine that an object of type ExitTile can be instantiated correctly.	
Maze class	Test that a maze can be drawn and updated. Test that the random generation of mazes produces solvable mazes. Test that player moves and updates trail correctly.	
Menus / Completed program	Test that each of the menus is displayed correctly and all of the menu elements function correctly (e.g. help button navigates to help menu 1).	

## Post development testing

This testing is much more formal than alpha testing. The post development tests are organised in a table in the order of the design objective(s), in order for the success of each to be evaluated. Exhaustive tests will use a variety of different input and test data in order to ensure the system follows correct procedures. Each of the objectives in the Design Objectives section of the *Nature of the solution* will have one or more corresponding tests to ensure that each objective is correctly addressed. The test table plan is included in this section so that the tests have been pre-emptively created before any development occurs. This allows for the tests to be rigorously planned in an attempt to ‘break’ the system and identify any flaws or errors when edge cases are provided as input.

Another test table in the testing section will then be created which will reference each test in this plan. Each test will be completed with references to evidence (including screenshots and / or textual explanations). Should any of the tests fail, modifications to the system will be made in order to rectify any failures. The tests will then be repeated in a condensed table and re-assessed for success.

**Format of the post development test table**

The proposed tests are arranged into the table on the following pages. These tests will be completed after software development and alpha testing. Below is a dissection and explanation of the columns in the test table:

- **No** – this is a unique test number that is used to identify each test. When testing is completed after software development, its unique testing number will be used to reference a test in the test strategy.
- **Objective Reference(s)** – Each test has one or more references to objectives in the Final Design Objectives section in the *Nature of the solution*. These are the objectives that the test will evaluate for success. Each design objective has at least one test associated with it. This ensures that the whole system will be rigorously tested.
- **Purpose of test** – this is a brief description of why the test is being carried out.
- **Method** – this is a set of instructions on how to carry out the test in the Testing section and how to reproduce the conditions that will cause the test to be evaluated.
- **Input / Test data** – this is input or data that is used in conjunction with the test to produce desired results. N/A denotes that no input or test data will be input into the system for this test (for example, this test may simply be to visually inspect something present on the webpage). Note that it is implied as an input to the system that the webpage hosting the game is visited in a web browser, although tests may state this anyway for the sake of clarity.
- **Type of input / test data** – This is split into three categories: normal, abnormal and extreme. Normal data is characteristic of data that should be accepted by the system. Abnormal data and extreme data are data that are not desirable although should be dealt with appropriately. Note that abnormal and extreme data is sparse in a game that accepts limited input and constrains the input to menu navigation and movement entry via the keyboard.
- **Expected result** – this is the desirable result that should be produced when the test is carried out. If this result is found to occur when the testing is carried out then test can be denoted as a success.

**Post development test table**

<b>Test No</b>	<b>Objective Reference(s)</b>	<b>Purpose of test</b>	<b>Method of test</b>	<b>Input / Test data</b>	<b>Type of input / test data</b>	<b>Expected Result</b>
1	1.a.i	Test that pressing the new game button on the main menu will start a new maze.	Press the new game button on the main menu. Observe and record evidence of result.	New game button	Normal	A new maze is displayed.
2	1.a.ii	Test that pressing the continue button, when disabled, does nothing.	Press the continue button when it is disabled. Observe and record evidence of result.	Continue button (must be disabled)	Abnormal	Nothing.
3	1.a.ii	Test that pressing the continue button, when enabled, loads a level from local storage and displays it.	Use the JavaScript console in web browser to enter command “localStorage” to return data to determine there is a saved level. Press the continue button on the main menu and observe and record whether the size of maze generated (if any) matches data in local storage.	Continue button	Normal	A maze is displayed that contains the width and height dimensions of the ones specified in the game’s local storage.
4	1.a.ii	Test that, when game data is saved in local storage, that pressing the continue button loads a maze of the correct size.	Ensure there is data saved in local storage by observing the output of the “localStorage” command when entered into the JavaScript console of a web browser. Then, press the continue button and observe whether generated maze matches dimensions of that in local storage.	Data in local storage. Continue game button.	Normal	The maze generated and displayed when the continue button is pressed will match the dimensions saved in local storage.
5	1.a.iii	Test that pressing the help button on the main menu will navigate to the first help menu.	Press the help button on the main menu. Observe and record evidence of result.	Help button	Normal	The first help menu is displayed.

6	1.a.iv	Test that pressing the back button on the help menus navigates to the main menu.	On the first, second and third help menus, press the each respective back button. Observe whether the main menu is displayed. Record evidence of result.	Back button (first, second and third help menus respectively)	Normal	For each of three back buttons, main menu is displayed.
7	1.a.v	Test that pressing the right arrow button on the first help menu navigates to the second help menu.	Press the right arrow button on the first help menu. Observe and record evidence of result.	Right arrow button (on first help menu)	Normal	The second help menu is displayed.
8	1.a.v	Test that pressing the right arrow button on the second help menu navigates to the third help menu.	Press the right arrow button on the second help menu. Observe and record evidence of result.	Right arrow button (on second help menu)	Normal	The third help menu is displayed.
9	1.a.v	Test that pressing on the disabled right arrow button on the third help menu does nothing.	Press the disabled right arrow button on the third help menu. Observe and record evidence of result.	Right arrow button (on third help menu)	Normal	Nothing.
10	1.a.vi	Test that pressing on the left arrow button on the third help menu navigates to the second help menu.	Press the left arrow button on the third help menu. Observe and record evidence of result.	Left arrow button (on third help menu)	Normal	The second help menu is displayed.
11	1.a.vi	Test that pressing on the left arrow button on the second help menu navigates to the first help menu.	Press the left arrow button on the second help menu. Observe and record evidence of result.	Left arrow button (on second help menu)	Normal	The first help menu is displayed.
12	1.a.vi	Test that pressing on the disabled left arrow button on the first help menu does nothing.	Press the disabled left arrow button on the first help menu. Observe and record evidence of result.	Left arrow button (on first help menu)	Normal	Nothing
13	1.a.vii	Test that when the quit to main menu on the end of level menu is pressed that is navigates to the main menu	Press the quit to main menu button on the end of level menu. Observe whether main menu is displayed and record evidence of result.	Quit to main menu button (on end of level menu)	Normal	The main menu is displayed.

14	1.a.vii	Test that pressing the quit to main menu button on the pause menu navigates to the main menu	When on the pause menu, press the quit to main menu button. Observe whether the main menu is displayed and record evidence of result.	Quit to main menu button (on pause menu)	Normal	Main menu is displayed.
15	1.a.viii	Test that when the next level button on the end of level menu is pressed that it displays a new maze.	Press the next level button on the end of level menu and observe and record evidence of result.	Next level button	Normal	A new maze is displayed.
16	1.b.i	Test that W key moves player up by one tile if not blocked by a wall	Press the W key on a maze where no wall blocks a tile above the player and observe whether player moves up one tile and record evidence of result.	W key	Normal	Player moves up one tile.
17	1.b.ii	Test that Up arrow key moves player up by one tile if not blocked by a wall	Press the up arrow key on a maze where no wall blocks a tile above the player and observe whether the player moves up one tile and record evidence of result.	Up arrow key	Normal	Player moves up one tile.
18	1.b.iii	Test that A key moves player left by one tile if not blocked by a wall	Press the A key on a maze where no wall blocks a tile left of the player and observe whether player moves left one tile and record evidence of result.	A key	Normal	Player moves left one tile.
19	1.b.iv	Test that Left arrow key moves player left by one tile if not blocked by a wall	Press the left arrow key on a maze where no wall blocks a tile left of the player and observe whether the player moves left one tile and record evidence of result.	Left arrow key	Normal	Player moves left one tile.
20	1.b.v	Test that S key moves player down by one tile if not blocked by a wall	Press the S key on a maze where no wall blocks a tile below the player and observe whether player moves down one tile and record evidence of result.	S key	Normal	Player moves down one tile.
21	1.b.vi	Test that Down arrow key moves player down by one tile if not blocked by a wall	Press the down arrow key on a maze where no wall blocks a tile below the player and observe whether the player moves down one tile and record evidence of result.	Down arrow key	Normal	Player moves down one tile.

22	1.b.vii	Test that D key moves player right by one tile if not blocked by a wall	Press the D key on a maze where no wall blocks a tile right of the player and observe whether the player moves right one tile and record evidence of result.	D key	Normal	Player moves right one tile.
23	1.b.viii	Test that Right key moves player right by one tile if not blocked by a wall	Press the right arrow key on a maze where no wall blocks a tile right of the player and observe whether the player moves right one tile and record evidence of result.	Right arrow key	Normal	Player moves right one tile.
24	1.b.ix	Test that pressing the escape key when a maze is displayed navigates to the pause menu	When a maze is currently displayed, press the escape key. Observe whether the pause menu is displayed and record evidence of result.	Escape key (when maze is displayed)	Normal	The pause menu is displayed.
25	1.b.ix	Test that pressing the escape key when on the pause menu returns to displaying the maze	When on the pause menu, press the escape key. Observe whether the maze is displayed.	Escape key (on pause menu)	Normal	Maze is displayed.
26	1.b.ix	Test that pressing the escape key when on the main menu does nothing.	When on the main menu, press the escape key. Observe whether nothing happens.	Escape key (on main menu)	Abnormal	Nothing.
27	1.b.ix	Test that pressing the escape key when on a help menu does nothing	When on any of the help menus, press the escape key. Observe whether nothing happens.	Escape key	Abnormal	Nothing.
28	2.a.i, 2.d.iv	Move the player to the exit tile of a 4x4 maze and ensure the maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Press the new game button. Move the player, using the arrow keys, to the exit tile, if possible.	Combination of arrow keys in order to reach exit tile.	Normal	The exit tile should be reachable and thus confirm the maze is solvable.
29	2.a.i, 2.d.iv	Check that the 6x6 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Press the new game button. Complete the first level of the game. Press the next level button on the end of level menu. Move the player, using the arrow keys, to the exit tile, if possible.	Combination of arrow keys in order to reach exit tile.	Normal	The exit tile should be reachable and thus confirm the maze is solvable.

30	2.a.i, 2.d.iv	Check that the 8x8 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Press the new game button. Complete the first two levels of the game. Press the next level button on the end of level menu. Move the player, using the arrow keys, to the exit tile, if possible.	Combination of arrow keys in order to reach exit tile.	Normal	The exit tile should be reachable and thus confirm the maze is solvable.
31	2.a.i, 2.d.iv	Check that the 10x10 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Press the new game button. Complete the first three levels of the game. Press the next level button on the end of level menu. Move the player, using the arrow keys, to the exit tile, if possible.	Combination of arrow keys in order to reach exit tile.	Normal	The exit tile should be reachable and thus confirm the maze is solvable.
32	2.a.ii	Test that a 4 by 4 tile maze is generated when the new game button on the main menu is pressed.	Press the new game button on the main menu. Observe that a maze is displayed and count the number of tiles present in the maze and record evidence of observation.	New game button on main menu.	Normal	A maze is generated and displayed that is 4 by 4 tiles in size (16 total tiles).
33	2.a.iii	Test that the exit tile and player are placed randomly in the maze at least a third of the maze apart from each other. Repeat this for multiple random mazes.	Visit the webpage where the game is hosted. Press the new game button.  Use a pixel ruler browser extension to measure the absolute pixel distance in the x and y directions between the exit tile and player tile. Then measure the dimensions of the maze in pixels. Divide the x and y distances between exit tile and player by the dimensions of the maze. Record whether this value is greater than 0.33.	New game button.	Normal	The player and exit tile are randomly placed in the maze, at least a third of the size of the maze apart.

34	2.a.iv, 3.l.xii	<p>Test that, for a maze larger than the dimensions of the game window (512 by 512 pixels), then the camera is centred on the player and follows the player, after panning has occurred.</p>	<p>Start a maze that is larger than the size of the game window (512 by 512 pixels). This can be achieved by playing through the game until a suitable level is reached.</p> <p>Use a pixel ruler browser extension in order to measure half of the game window. This can then be used to determine if the player is in the centre of the viewport. Use arbitrary movement input keys to determine if player is still at centre of camera. Record evidence.</p>	<i>Input required reaching suitable level larger than game window.</i>  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player).	Normal	The camera is centred on the player and follows the player when the player moves.
35	2.a.v	<p>Test that, for a maze larger than the dimensions of the game window (512 by 512 pixels), the camera begins the level centred on the exit tile then pans over to the player.</p>	<p>Start a maze that is larger than the size of the game window (512 by 512 pixels). This can be achieved by playing through the game until a suitable level is reached.</p> <p>Observe and take evidence of the whether the camera pans from exit tile to player.</p>	<i>Input required reaching suitable level larger than game window.</i>	Normal	The camera should pan from the exit tile to the player just after starting maze that cannot fit wholly in the game window.

36	2.b	<p>Test that every subsequent maze that is generated is 2 tiles larger in the horizontal and vertical dimensions.</p>	<p>Start a maze with the new game button. Record the size of the first level maze (should be 4 by 4). Complete the maze then proceed to the next make and record size. Repeat procedure up to level 4. Record evidence of each maze size and observe whether there is an increase of 2 tiles in each dimension as the level increases.</p>	<p>New game button Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to complete maze).</p> <p>Next level button.</p> <p><i>Last two sets of input are repeated for 4 levels.</i></p>	Normal	<p>Subsequent mazes that are generated should be 2 tiles larger in the horizontal and vertical dimensions than the previous level.</p>
37	2.c	<p>Test that, if a game is saved in local storage and local storage is deleted, the game disables the continue button</p>	<p>Enter the JavaScript console and enter the command “localStorage.clear()” in order to delete the contents of localStorage. Type “localStorage” to confirm it is empty. Next, refresh the browser page and then observe and record evidence of whether the continue button is disabled.</p>	<p>Delete the contents of local storage in the JavaScript console.</p> <p>Visit the webpage.</p>	Extreme	<p>When local storage is deleted the continue button should be disabled.</p>

38	2.c	Test that, when quitting to the main menu from the next level menu, that the continue button is enabled and the game can be loaded	When on the end of level menu, press the quit to main menu button. Ensure there is data saved in local storage by observing the output of the “localStorage” command when entered into the JavaScript console of a web browser. Observe whether the continue button is not disabled and press it if it is not. Observe whether the maze matches the dimensions saved in local storage.	Quit to main menu button (on end of level menu).  Continue button.	Normal	The continue button should not be disabled and when pressed should generate and display a maze of dimensions saved in local storage.
39	2.c	Test that, when quitting to the main menu from the pause menu, that the continue button is enabled and the game can be loaded.	Press the escape key when a maze greater than 4 by 4 dimensions is displayed to enter the pause menu. Press the quit to main menu button. Ensure there is data saved in local storage by observing the output of the “localStorage” command when entered into the JavaScript console of a web browser. Observe whether the continue button is not disabled and press it if it is not. Observe whether the maze matches the dimensions saved in local storage.	Quit to main menu button (on end of level menu).  Continue button.	Normal	The continue button should not be disabled and when pressed should generate and display a maze of dimensions saved in local storage.
40	2.d.i	Test that pressing W key does not move player up by one tile when the path is blocked by a wall	Press the W key on a maze where a wall blocks a tile above the player and observe whether the player stays on the same tile and record evidence of result.	W key	Abnormal	Player stays on the same tile.
41	2.d.i	Test that pressing A key does not move player left by one tile when the path is blocked by a wall	Press the A key on a maze where a wall blocks a tile left of the player and observe whether the player stays on the same tile and record evidence of result.	A key	Abnormal	Player stays on the same tile.

42	2.d.i	Test that pressing S key does not move player down by one tile when the path is blocked by a wall	Press the S key on a maze where a wall blocks a tile below the player and observe whether the player stays on the same tile and record evidence of result.	S key	Abnormal	Player stays on the same tile.
43	2.d.i	Test that pressing D key does not move player right by one tile when the path is blocked by a wall	Press the D key on a maze where no wall blocks a tile right of the player and observe whether the player stays on the same tile and record evidence of result.	D key	Abnormal	Player stays on the same tile.
44	2.d.i	Test that pressing Up arrow key does not move player up by one tile when the path is blocked by a wall	Press the up arrow key on a maze where a wall blocks a tile above the player and observe whether the player stays on the same tile and record evidence of result.	Up arrow key	Abnormal	Player stays on the same tile.
45	2.d.i	Test that pressing Left arrow key does not move player left by one tile when the path is blocked by a wall	Press the left arrow key on a maze where a wall blocks a tile left of the player and observe whether the player stays on the same tile and record evidence of result.	Left arrow key	Abnormal	Player stays on the same tile.
46	2.d.i	Test that pressing Down arrow key does not move player down by one tile when the path is blocked by a wall	Press the down arrow key on a maze where a wall blocks a tile below the player and observe whether the player stays on the same tile and record evidence of result.	Down arrow key	Abnormal	Player stays on the same tile.
47	2.d.i	Test that pressing Right arrow key does not move player right by one tile when the path is blocked by a wall	Press the right arrow key on a maze where no wall blocks a tile right of the player and observe whether the player stays on the same tile and record evidence of result.	Right arrow key	Abnormal	Player stays on the same tile.
48	2.d.ii	Test that pressing W key does not move player up by one tile when the player is on the top edge of the maze.	Press the W key on a maze where the player is situated on the top edge of the maze and observe whether the player stays on the same tile and record evidence of result.	W key	Extreme	Player stays on the same tile.
49	2.d.ii	Test that pressing A key does not move player left by one tile when the player is on the left edge of the maze.	Press the A key on a maze where the player is situated on the left edge of the maze and observe whether the player stays on the same tile and record evidence of result.	A key	Extreme	Player stays on the same tile.

50	2.d.ii	Test that pressing S key does not move player down by one tile when the player is on the bottom edge of the maze.	Press the S key on a maze where the player is situated on the bottom edge of the maze and observe whether the player stays on the same tile and record evidence of result.	S key	Extreme	Player stays on the same tile.
51	2.d.ii	Test that pressing D key does not move player right by one tile when the player is on the right edge of the maze.	Press the D key on a maze where the player is situated on the right edge of the maze and observe whether the player stays on the same tile and record evidence of result.	D key	Extreme	Player stays on the same tile.
52	2.d.ii	Test that pressing Up arrow key does not move player up by one tile when the player is on the top edge of the maze.	Press the up arrow key on a maze where the player is situated on the top edge of the maze and observe whether the player stays on the same tile and record evidence of result.	Up arrow key	Extreme	Player stays on the same tile.
53	2.d.ii	Test that pressing Left arrow key does not move player left by one tile when the player is on the left edge of the maze.	Press the left arrow key on a maze where the player is situated on the left edge of the maze and observe whether the player stays on the same tile and record evidence of result.	Left arrow key	Extreme	Player stays on the same tile.
54	2.d.ii	Test that pressing Down arrow key does not move player down by one tile when the player is on the bottom edge of the maze.	Press the bottom arrow key on a maze where the player is situated on the bottom edge of the maze and observe whether the player stays on the same tile and record evidence of result.	Down arrow key	Extreme	Player stays on the same tile.
55	2.d.ii	Test that pressing Right arrow key does not move player right by one tile when the player is on the right edge of the maze.	Press the right arrow key on a maze where the player is situated on the right edge of the maze and observe whether the player stays on the same tile and record evidence of result.	Right arrow key	Extreme	Player stays on the same tile.

56	2.d.iii	Check that pressing the W, A, S, D, Up arrow, Right arrow, Down arrow and Left arrow keys when on the main menu does not cause the player to move tiles in the maze.	Press an arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow keys when on the main menu. Start a new maze with new game button and observe whether any movement has occurred by the player (by observing presence of a trail). Record evidence of result.	Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow.  New game button.	Abnormal	Upon displaying of maze the player has not performed any movement.
57	2.d.iii	Check that pressing the W, A, S, D, Up arrow, Right arrow, Down arrow and Left arrow keys when on the pause menu does not cause the player to move tiles in the maze.	Press an arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow keys when on the main menu. Resume maze by pressing escape key and observe whether any movement has occurred by the player (by observing presence of a trail). Record evidence of result.	Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow.  Escape key.	Extreme	Upon displaying of maze the player has not performed any movement.
58	2.d.iii	Check that pressing the W, A, S, D, Up arrow, Right arrow, Down arrow and Left arrow keys when on the end of level menu does not cause the player to move tiles in the maze.	Press an arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow keys when on the end of level menu. Start the next maze with next level button and observe whether any movement has occurred by the player (by observing presence of a trail). Record evidence of result.	Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow.  Next level button.	Extreme	Upon displaying of maze the player has not performed any movement.

59	2.d.v, 3.l.xiv	Test that the player leaves a trail correctly.	Press the new game button. Use movement keys to move player around the maze. Observe whether the player leaves a coloured trail and record evidence of result. Repeat for multiple mazes.	New game button  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player).	Normal	The player should leave a trail on all the tiles that it moves over.
60	2.d.vi, 3.l.xiii	Test that the colour of the player fades correctly over time. The player should fade from green to purple and then back again in a recurring cycle.	Start a new game by pressing the new game button. This will display a new maze. Take screenshots of maze every 10 seconds. Observe and record evidence of whether the colour of the player fades between the correct hue values.	New game button.	Normal	The player should fade from green to purple and then back again in a recurring cycle.
61	3.a	Test that the game is embedded in a single web page.	Visit the webpage. Inspect and record whether a webpage that contains an embedded game is rendered	N/A	Normal	The game should be embedded in a single webpage
62	3.b.i, 3.b.ii, 3.b.iii, 3.b.iv, 3.l.i	Test to ensure that there is a heading, subheading, canvas and footer present on the webpage.	Observe and record evidence of the web page on which the game is displayed and inspect whether the necessary elements are displayed. Note that the element inspector in the web browser will need to be used to confirm the presence of a canvas element.	N/A	Normal	The web page should contain: <ul style="list-style-type: none"> <li>• A header with the name of the game (Mazr).</li> <li>• A subheading that explains the game briefly.</li> <li>• The game should be embedded in a HTML canvas.</li> <li>• A footer that contains a link to the user documentation.</li> </ul>

63	3.c	Test that when JavaScript is disabled in the user's browser, then a suitable error message is displayed and none of the menus are displayed.	Go into the browser's settings and disable JavaScript. Observe and record evidence of whether the error message is displayed and that no menus are displayed.	N/A	Extreme	A suitable error message should be displayed when the user has JavaScript turned off in their browser. No menus should be displayed.
64	3.d	Test that The game will be composed of a set of menus and each will be contained in div HTML elements and all appear on the same page although hidden / unhidden to simulate multiple pages being navigated.	Go into the element inspector in the web browser and use it to check that each of the menus is contained in the HTML. Use the tool to check that only one is visible at a given time. Observe and record evidence of result.	N/A	Normal	The game should be composed of a set of menus and each will be contained within a HTML div element. Only one menu should be visible at a given time.
65	3.e.i, 3.e.ii, 3.e.iii	Test that the main menu contains correct content.	On the main menu observe and record whether all the necessary content is displayed.	N/A	Normal	<p>The main menu should contain:</p> <ul style="list-style-type: none"> <li>• A button to continue with a game saved in local storage. This will be displayed if no game is stored in local storage.</li> <li>• A button to start a new game.</li> <li>• A button to go to the first help menu.</li> </ul>

66	3.f.i, 3.f.ii, 3.f.iii, 3.g.i, 3.g.ii	Test that the first help menu contains the correct content.	On the first help menu observe and record whether all the necessary content is displayed.	N/A	Normal	<p>All help menus (and thus the first help menu) should contain:</p> <ul style="list-style-type: none"><li>• A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.</li><li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li><li>• A back button to return to the main menu from any of the help menus.</li></ul> <p>The first help menu should contain:</p> <ul style="list-style-type: none"><li>• Onscreen help that explains the format of the game and the goal of the game.</li><li>• An image of a small maze with a player and exit tile.</li></ul>
----	---	---	---	-----	--------	--

67	3.f.i, 3.f.ii, 3.f.iii, 3.h.i, 3.h.ii	Test that the second help menu contains the correct content.	On the second help menu observe and record whether all the necessary content is displayed.	N/A	Normal	<p>All help menus (and thus the second help menu) should contain:</p> <ul style="list-style-type: none"> <li>• A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.</li> <li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li> <li>• A back button to return to the main menu from any of the help menus.</li> </ul> <p>The second help menu should contain:</p> <ul style="list-style-type: none"> <li>• Onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.</li> <li>• Image of a small maze where the player is leaving a coloured trail moving to the exit tile.</li> </ul>
----	---	--	--	-----	--------	---

68	3.f.i, 3.f.ii, 3.f.iii, 3.i.i, 3.i.ii	Test that the third help menu contains the correct content.	On the third help menu observe and record whether all the necessary content is displayed.	N/A	Normal	<p>All help menus (and thus the third help menu) should contain:</p> <ul style="list-style-type: none"> <li>• A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.</li> <li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li> <li>• A back button to return to the main menu from any of the help menus.</li> </ul> <p>The third help menu should contain:</p> <ul style="list-style-type: none"> <li>• Onscreen help that explains how completing a maze subsequently generates a larger maze for the player to solve.</li> <li>• Image of a larger maze.</li> </ul>
----	---	---	---	-----	--------	--

69	3.j.i, 3.j.ii, 3.j.iii, 3.j.iv, 3.j.v	Test that the end of level menu contains the correct content.	On the end of level menu observe and record whether all the necessary content is displayed.	N/A	Normal	<p>The end of level menu should contain:</p> <ul style="list-style-type: none"><li>• Text congratulating the user on their completion of the maze.</li><li>• Text that contains the time taken for the user to complete the maze.</li><li>• Text that contains the number of steps taken for the user to complete the maze.</li><li>• A button that allows the user to go to the next maze.</li><li>• A button that allows the user to quit to the main menu.</li></ul>
----	---	---	---	-----	--------	---

70	3.k.i, 3.k.ii	Test that the pause menu contains the correct content.	On the pause menu observe and record whether all the necessary content is displayed.	N/A	Normal	The pause menu should contain: <ul style="list-style-type: none"><li>• A button that allows the user to quit to the main menu.</li></ul>
71	3.l.ii	Check that the game canvas is 512 by 512 pixels	Visit the webpage where the game is hosted. Press the new game button.  Use a pixel ruler browser extension to measure the pixel dimensions of the game screen and record evidence.	New game button.	Normal	The screen should be 512px by 512px.

72	3.l.iii	Test the maze is drawn to the canvas as a grid of tiles with walls, along with an exit tile and a player.	Press the new game button. Inspect whether the maze contains a grid of tiles, each with a set of walls and an exit tile and player. Record evidence of result.	New game button	Normal	A grid of tiles that represent the maze must be drawn to the canvas, along with the exit tile and player.
73	3.l.iv	Check that the tiles in the maze are 32 by 32 pixels	Visit the webpage where the game is hosted. Press the new game button.  Use a pixel ruler browser extension to measure 3 random tiles in each maze. Repeat for 3 mazes of different sizes. Record evidence of the size of these samples and observe whether they are 32 by 32 pixels.	New game button.	Normal	The tile should be 32px by 32px.
74	3.l.v	Check that the player in the maze is 32 by 32 pixels	Press the new game button to generate a new maze.  Use a pixel ruler browser extension to measure the pixel dimensions of the player. Record evidence of result.	New game button.	Normal	The player should be 32px by 32px.
75	3.l.vi	Check that the exit tile in the maze is 32 by 32 pixels	Press the new game button to generate a new maze.  Use a pixel ruler browser extension to measure the pixel dimensions of the exit tile. Record evidence of result.	New game button.	Normal	The exit tile should be 32px by 32px.

76	3.l.vii	Check that the walls in the maze are 32 by 2 pixels	<p>Press the new game button to generate a new maze.</p> <p>Use a pixel ruler browser extension to measure the pixel dimensions of a wall that is one tile long. Record evidence of result.</p>	New game button.	Normal	The walls of the maze should be 32 by 2 pixels.
77	3.l.viii	Test that The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of grey.	<p>Press the new game button to start a new maze.</p> <p>Inspect and take evidence of the maze and the colour of the tiles in each maze.</p> <p>Progress through the levels up to level 6 and inspect and take evidence of the maze in each. Record whether the tiles in the maze are assorted shades of grey.</p>	<p>New game button</p> <p><i>Input required to reach 6 different levels in order to check shades of grey.</i></p> <p>Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player) and next level button repeatedly.</p>	Normal	The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of grey.

78	3.l. ix	Test that the saturation and lightness values of the player remain constant at 80% and 40% respectively.	Press the new game button to start a new game. Take 6 screenshots over 30 second intervals and import into an image editing application with an eyedropper facility to inspect the colour of the player. Record whether the saturation and lightness remain at the correct constant values for each of the interval samples.	New game button	Normal	The player should have a constant saturation of 80% and a constant lightness of 40%.
79	3.l. x	Test that the colour of the exit tile is #FF8000	Press the new game button to start a new maze. Take a screenshot and use an image editing application with an eyedropper facility to inspect the colour of the exit tile. Record the result along with evidence.	New game button	Normal	The exit tile should have the colour #FF8000
80	3.l. xi	Test that, for a maze smaller than the dimensions of the game window (512 by 512 pixels), then the maze is fixed in the centre of the window and the camera is disabled.	Press the new game button to start a new maze.  Record and observe whether the camera is fixed on the centre of the player using a pixel ruler browser extension. Record and observe whether the camera follows the player around the maze.	New game button.	Normal	When the maze does not fit in the game window (512 by 512 pixels) then the camera should be enabled and follow the player around the maze.

## End user testing

This will make use of the sample of five end users. They will each have time to use the developed system and each will answer a questionnaire. The data collected from this questionnaire will result in possible modifications to the system – in order to resolve bugs or errors they have found or indeed change areas of the system to better suit their opinions.

Below is the test that will be used to record the response to the end user testing. Below is a description of each of the column headings:

- **Question no** – Unique number for each question in end user testing questionnaire.
- **Question** – The question that will be asked of the user.
- **Purpose of question** – The reason why the question is being asked. Description of the information that is being ascertained.
- **Frequency of response of users (Y/N)** – This is a record of the number of Yes / No answers to each of the questions. Split into two columns – one to record the frequency of ‘Yes’ answers and one to record the frequency of ‘No’ answers. Note that both frequencies should sum together to five for each question.
- **Additional comments (optional)** – This is where any additional comments, which users may wish to make regarding their answer to the questions.

Question no	Question	Purpose of question	Frequency of response of users (Y/N)		Additional comments (optional)
			Yes	No	
1	Was the game easy to access via your web browser?	To gain an understanding of whether the game is accessible effectively in the end user’s web browsers.			
2	Were the menus easy to use and understand?	Ascertain knowledge of whether the layout and content of the menus was easy to understand for the end users.			
3	Were the onscreen help menus sufficient in understanding how the game works?	To gain an understanding of whether the onscreen help menus that were integrated into the game contain sufficient information in order to play the game.			
4	Was the system easy to use?	Ascertain knowledge of whether the end users find the game approachable and easy to use.			
5	Were the controls easy to use and suitable?	To gain an understanding as to whether the controls were natural for the end users or whether they had any trouble using them.			
6	Did the mazes provide a suitable challenge?	To understand whether the game provides a suitable challenge without being too easy or too difficult. Assessment of learning curve.			

7	Did you like the colour style and graphics used in the maze?	To ascertain whether the users like or dislike the ' <i>modern, blocky</i> ' style and graphics that are used in the game.			
8	Did you discover any bugs or inconsistencies in the game? (Provide details in comments)	To find out whether the users encountered any issues such as bugs or unexpected errors whilst playing the game.			
9	Did the progress of the game save and load correctly?	To gain an understanding of whether the end users found that the saving and loading features worked when they played the game.			
10	Any further comments about potential modifications?	Enquire as to whether the end users had any further comments to make about the system and whether they have any ideas for modifications.			

## Acceptance testing

Acceptance testing will involve the client signing off on the new system as being thoroughly tested and without any noticeable or foreseeable faults. This involves the client firstly using the new system by herself as an end user. Then, the client will review multiple documents that represent the evidence of testing at each stage in the development.

Firstly, the client will be presented with the alpha test table containing references to pages on which alpha testing is completed during the *Software development* stage of the project. Next, the client will be presented with the post-development testing in the form of the test table. This will allow the client to review whether they believe that each of the design objectives is successful (or whether they were successful after modification). The client will have the chance to view the end user testing questionnaire and are happy that user involvement was taken into account in form of any further modifications.

An interview will then be arranged with the client in order to discuss their use of the system. Alpha testing, post-development testing and end user testing will also be reviewed. Finally, the client will possibly suggest any final modifications they may feel necessary and then, providing evidence of these possible modifications is presented, the client will sign off the testing.

# Software Development

## Introduction

This section documents the development process that occurred as Mazr was created. It is presented as iterative development process in which a record is made of all changes to all files one by one in order to build up a codebase that will encompass the final system.

## Technical Description

The system will be written using a combination of HTML markup, CSS stylesheets and JavaScript code files. The HTML and CSS will be used to design and render the content and style of the webpage whilst the JavaScript will be used to control the logic and dynamic elements, for example drawing and updating the maze and its elements. As JavaScript is a scripting language no compilation will be necessary at any stage, as the web browser will execute the scripts that are written when the browser renders the webpage.

The Sublime Text IDE (Integrated development environment) will be used to write all of the files for the game. It offers simple syntax highlighting and a file tree manager that allows the development environment to be viewed and manipulated. Sublime Text has no native ability to compile code but this does not matter, as no compilation is necessary. Sublime Text is perfect for the scope of this project. The Google Chrome web browser will be used to render the local HTML file, for purposes of alpha testing and then further testing in the future.

Note that the development environment will consist of a HTML webpage that will contain links to all of the project's CSS, JavaScript and any other file constituents. This will all be contained within a single directory. Loading the game is as simple as opening the local HTML file in a supported web browser (see the Requirements specification in the Investigation and analysis section). When the project is completed this development environment can equally be used as a production environment and placed on a server for deployment (where the HTML can be accessed via public IP address or registered domain name).

The solution will use and relate to the design specification as it is developed. The design specification, including all user-agreed elements will need to appear in the development of the system. Below is a list of points of how the design specification and other related design documents must be used in the software development.

- The Design Objectives will be used as a guide to ensure that all elements are developed correctly. This contains the specification that will be evaluated at the system Testing stage and so must be used a guide for development at this point.
- The screen layouts will be used to write the HTML content that will appear in each of the menus on the webpage.
- The CSS styling table will be used to define all of the Cascading Style Sheet styles that will influence the design of the website.
- The Data Structures will be used in order to ensure that all of the correct attributes, methods, variables and functions are included for each class function and also in the global program scope.
- The Algorithms section will be used to write each of the algorithms that will appear as methods and class functions in the new system.
- The Alpha-testing plan at the beginning of the Test strategy will be used as a preliminary guide on how the alpha testing will be completed at each of the iterative stages of the software development.
- The Post development test table in the Test Strategy will be used to evaluate each of the design objectives for success once the software is completed.

The requirements specification is addressed at each stage of the development process, with relevant requirements being referenced at each appropriate stage. These notes appear under section headings and are italicised.

### **Alpha Testing**

This following section on the software development contains intermittent alpha testing. This involves either manipulation of the JavaScript console of a web browser (in this case Google Chrome) and observing the outputs or writing some additional JavaScript in a file in order to produce some desirable test effect (such as drawing on the HTML canvas). In either case, this alpha testing is labelled and test code is clearly denoted. Note there is a more rigorous testing of the solution in the next section. The Alpha testing will loosely follow the plan defined in the Test Strategy and proof it has been included will be provided in the Testing section in the form of page references to this section.

### **Modularisation of code**

In the Nature of the solution section of the report, the top down design paradigm was discussed as the method that would be used in order to divide the system into smaller and smaller sections that are independent of each other and can be encapsulated within a single function, procedure or object. This is how the system will be modularised. The OO (object oriented) paradigm will be implemented to ensure that each component of the game is designed independently and then integrated at a later stage. This makes the code both easier to test (during the alpha testing stage) and easier to modify areas (as only small areas of the system will need to be searched and subsequently modified). It also means that should the system ever require an incremental update that a different programmer could easily understand the architecture of the modules and quickly edit or replace a given function or object.

Two libraries are also being used in the development of this project. The first is the jQuery JavaScript library that will be used to handle client interactions in the form of events that can trigger code to execute. The second is a CSS layout library that will normalise default style settings across browsers so that the webpage will be rendered more similarly across target browsers. Further information on these two libraries can be found in the Nature of the solution.

### **Annotation of code**

Rather than the code being annotated on the report directly, the code will be annotated with comments as the scripts are written and developed. This will ensure that very expressive and useful annotations are included in the code. Furthermore, it would allow another program to work with annotations that are included directly in the code. Therefore, annotated code can be viewed in the iterative process of the software development that follows or the full listing of the code at the end of the software development. Note that as the HTML and CSS is not technically programming or scripting that comments are not included in these files. Think of these as the forms of a traditional window-based application (which would not include comments by the programmer at any point).

### **Full listing of HTML, CSS and JavaScript**

Provided at the end of this section is a complete listing of the entire markup, styling and JavaScript files that make up the solution. Each section is clearly labelled with the filename and contains an exact copy of the code as it is at the end of this software development section. Note this code is annotated with comments and subject to change as per the results of testing in the next section in order to remedy bugs.

### **Note on terminology**

JavaScript is an OO (object oriented) programming language and as such should have the ability to define classes from which objects can be instantiated. However, there is no explicit way to define a class in JavaScript – instead object literals can be created or functions can be used as classes to instantiate objects can be used. The functional method of class creation will be used heavily in the following development of the software so has been given a special term in the report – a class

function. This is to differentiate it from ordinary functions. Whenever the term class function is mentioned, it can be read as a function from which objects can be instantiated. This is to help differentiate from traditional functions and methods (functions that belong to an object).

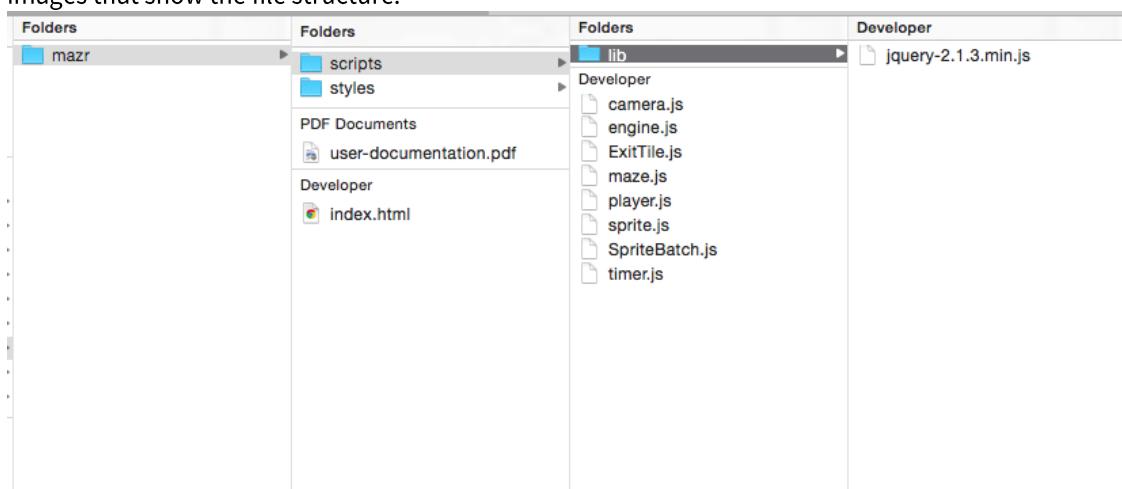
### Setting up the development environment

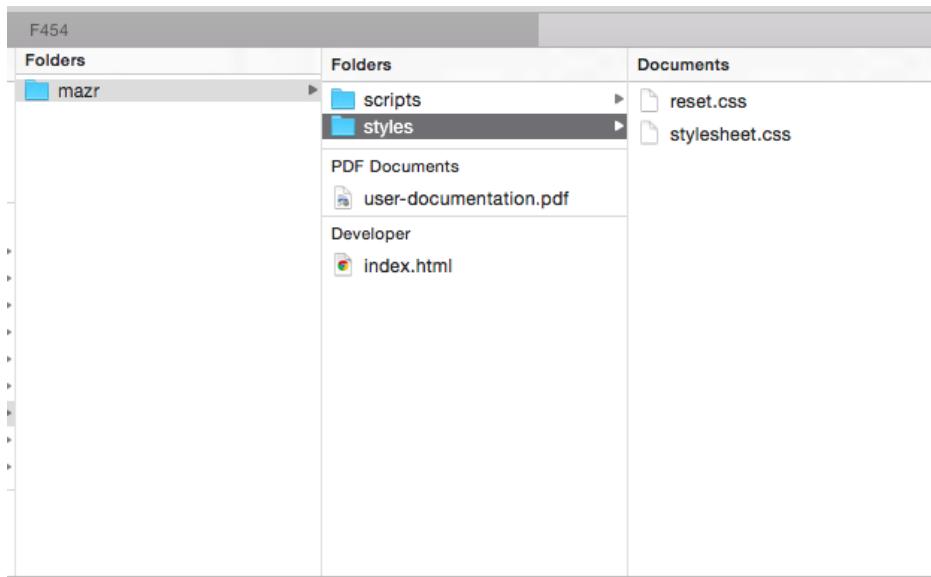
Before any of the scripts or HTML / CSS files can be written, a development environment needs to be set up. This involves the creation of a folder and file structure where all of the game assets will be placed. Mazr will be served to web browsers via HTML. Therefore, a single HTML document will be used as the webpage. During the software development and any form of testing this can be opened locally in a web browser. All of the other files are brought together and served as links to the HTML document.

A ‘mazr’ folder is created and an empty HTML file with the name index.html file is placed in this folder. It is called index.html because when a HTTP request for a HTML file is made to a web server and only a domain is supplied (for example <http://www.example.com>) then an index.html in the root directory of the server will be served and returned. Therefore, when the game’s files are deployed on a server this will naturally mean the game’s webpage will be returned when the domain is requested.

A ‘scripts’ folder is also created in the mazr folder. Empty script files are placed in this folder – camera.js, sprite.js, SpriteBatch.js, timer.js, player.js, ExitTile.js, maze.js and engine.js. These will contain all of the scripts that will make up the game logic. A ‘lib’ folder is created in the ‘scripts’ folder and will contain the jQuery library (see use of libraries in The Nature of The Solution section). This is downloaded from the jQuery website (The file was downloaded at the link <http://code.jquery.com/jquery-2.1.3.js> - note that the jQuery version may have been updated or changed since it was downloaded for use in Mazr) and placed in this folder.

In the Mazr folder, a styles folder is added. Into this two files are added. An empty cascading style sheet file is added to the file with the name stylesheet.css – this will contain the styling for the webpage. The Reset.css file mentioned in the use of libraries in the Nature of The Solution is also added to this folder. This has the effect of normalising the display of content in various browsers. (This was downloaded from the link <http://meyerweb.com/eric/tools/css/reset/reset.css> and may have been updated or changed since it was downloaded for use in Mazr). A temporary user-documentation.pdf file is created and placed in the mazr folder. This will be replaced with the actual user documentation file when it is created in the User Documentation section. Below are images that show the file structure.





With the development environment ready, the files can now start being populated in order to develop Mazr, the new system.

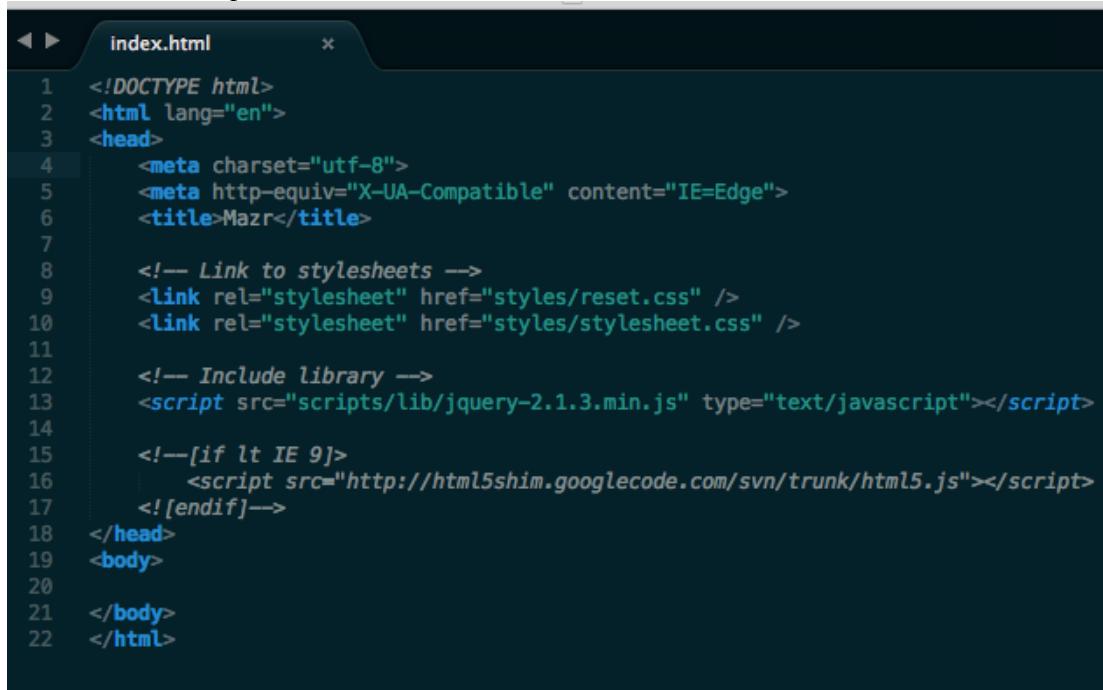
## Iterative development process

### HTML

This section pertains to the development of Output Requirements 19, 20, 21, 22, 23, 24 and 25 of the Requirements Specification.

HTML and CSS are not considered programming languages but instead methods to describe the contents, style and layout of a webpage. Therefore, as these are simple files that contain basic declarations of content and style there is no strict comments or annotations. This is reserved for the JavaScript programming, which may require additional description or understanding more than what is provided by the code. The index.html file is the first to be written. This file will include the content that will appear on the webpage that displays the game.

First, the skeleton layout of the document is created. Line one contains a document type tag that denotes it is a HTML 5 document. Line 2 and line 22 (note the closing tag will be pushed down as more content is added) contain opening and closing HTML tags between which the document's contents is enclosed. This is split between the head and body tags. The head tag contains two meta tags, on lines 4 and 5, the first of which denotes the character set required and the second tells Internet Explorer browsers to render it in the most advanced render mode the browser can provide. Lines 9 and 10 contain link tags that fetch the CSS files from the styles folder. The stylesheet.css, which will be populated with styling later, is linked to and the Reset.css library file downloaded from the Internet is linked to. Next, a script tag is used to pull the jQuery library placed in the lib folder in scripts in the last section. Finally, between lines 15 and 17 is a conditional comment that targets versions of Internet Explorer less than version 9. The HTML is only rendered between these comments in these browsers and fetches a JavaScript script file from an external source. This is a HTML 5 Shim that allows browser elements in HTML 5 to be displayed in the browser. This is shown in the file in the image below.



The screenshot shows a code editor window titled "index.html". The code is a standard HTML5 document structure. It starts with a doctype declaration, followed by an html tag with lang="en". Inside the html tag is a head section containing meta tags for charset and http-equiv, and a title tag with the value "Mazr". Below the head is a comment block starting with "Link to stylesheets" containing two link tags for "reset.css" and "stylesheet.css". A comment block for "Include library" contains a script tag for "jquery-2.1.3.min.js". A conditional comment for Internet Explorer versions less than 9 is present, pointing to "html5shim.googlecode.com/svn/trunk/html5.js". The document concludes with a closing head tag, a body tag, and a closing html tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<title>Mazr</title>
<!-- Link to stylesheets -->
<link rel="stylesheet" href="styles/reset.css" />
<link rel="stylesheet" href="styles/stylesheet.css" />
<!-- Include library -->
<script src="scripts/lib/jquery-2.1.3.min.js" type="text/javascript"></script>
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
</head>
<body>
</body>
</html>
```

Next, the body tag is initially populated. A h1 header and a paragraph are nested in a hgroup tag, which contains the game's header contents. Next a noscript tag contains an error message that will only be displayed if JavaScript is disabled in the browser. Following this, nested within a section tag, is a div with the class canvas-wrapper. Within this tag will appear the canvas, with an id of canvas, also with the various menus in the game. Each of the menus and the canvas are given the class of page (to distinguish between the menus and the canvas but providing styling for both). Following the closing section tag is a footer, which contains an "a" tag with a link to the user documentation, which is currently a temporary PDF file. This is shown in the image below.

```

19  <body>
20    <hgroup>
21      <h1>Mazr</h1>
22      <p>The endless maze solving puzzle game</p>
23    </hgroup>
24
25    <noscript>
26      You need to enable JavaScript in your browser in order to play the game.
27    </noscript>
28
29    <section>
30      <div class="canvas-wrapper">
31        <canvas id="canvas" class="page">
32          Unfortunately, your browser won't run Mazr. Please see our list of
33          compatible browsers.
34        </canvas>
35      </div>
36    </section>
37
38    <footer>
39      <a href="user-documentation.pdf">User guide</a>
40    </footer>
41
42  </body>

```

After the closing canvas tag, the div with the class main-menu and page is created. This contains all the content of the main menu – including the Continue, New game and Help buttons. Note each has a memorable class name in order to ensure that it can be referenced in the script files. This is shown in the image below.

```

35      <div class="main-menu page">
36        <div class="buttons">
37          <button tabindex="1" class="btn-continue">Continue</button>
38          <button tabindex="2" class="btn-new-game">New game<br>
39            <p class="under-text">Saved data will be erased</p>
40          <button tabindex="3" class="btn-help">Help</button>
41        </div>
42      </div>
43

```

After the main menu div, the first help menu div is created, with the classes help-1 and page. The help content is placed into the div in the form of paragraph tags and an image link. Then, a div containing two arrow buttons is placed in the help menu. These are given codes to arrow characters, which point in the correct directions (&#2190; is a left arrow character and &#8594; is a right arrow character). A back button is also written into the help menu and shown in the image below.

```

45      <div class="help-1 page">
46        <div class="buttons">
47          <h2>Help</h2><br>
48
49          <p>You are the coloured square. </p><br>
50          <p>
51            You are placed in a randomly generated maze.
52            All mazes are solvable. Move onto the orange square to
53            complete the maze.
54          </p><br>
55
56          
57
58          <div class="arrows">
59            <button disabled class="btn-arrow left">&#x2190;</button>
60            <button class="btn-arrow right">&#8594;</button>
61          </div>
62
63          <button class="btn-back">Back</button>
64        </div>

```

The second help menu is structured very similarly, with help content in the form of paragraph tags and an image and the two arrow buttons and a back button. Note there is no disabled attribute on either of the arrow button tags that suggest they both have a navigational purpose (in order to navigate to the other two menus). This is shown in the image below.

```

66      <div class="help-2 page">
67          <div class="buttons">
68              <h2>Help</h2><br>
69
70              <p>
71                  You can move around the maze with the arrow keys, providing
72                  no walls block your path.
73                  You will leave a coloured trail.
74              </p><br>
75              
76
77              <div class="arrows">
78                  <button class="btn-arrow left">⬅</button>
79                  <button class="btn-arrow right">➡</button>
80              </div>
81
82              <button class="btn-back">Back</button>
83          </div>
84      </div>

```

The third help menu is structured again like the second and first help menu. The right arrow button is disabled in this case and the help content is provided in the form of paragraph tags and an image tag. This is shown in the image below.

```

86 ▼      <div class="help-3 page">
87 ▼          <div class="buttons">
88              <h2>Help</h2><br>
89
90              <p>
91                  When you complete a maze, a new, larger maze is generated.
92              </p><br>
93
94              
95
96 ▼          <div class="arrows">
97              <button class="btn-arrow left">⬅</button>
98              <button disabled class="btn-arrow right">➡</button>
99          </div>
100
101          <button class="btn-back">Back</button>
102      </div>
103  </div>

```

The pause menu is quite simple, containing some information in a h2 and p tag along with a button that allows quitting to the main menu, as shown in the image below.

```

105      <div class="pause-menu page">
106          <div class="buttons">
107              <h2>Paused</h2>
108              <p>Press escape to return to the maze</p>
109              <button class="btn-quit">Quit to main menu</button>
110          </div>
111      </div>

```

The end of level menu is written to contain the congratulatory h2 heading tag, along with three p tags. The first contains the time required to complete the maze. Note that because the time must be determined programmatically by JavaScript code the area where the time must be inserted is instead occupied by a span tag, with the id time. This way, the script can specifically reference the element with id time and insert the time in there. The second paragraph tag has a similar theme, where a span with id steps will be used to reference the position where the number of steps taken

by the player needs to be recorded. The third paragraph tag contains text that says that the user's progress has been saved. Finally, two buttons are included, one to move the user to the next level and one to quit and navigate the user to the main menu. This is shown in the image below.

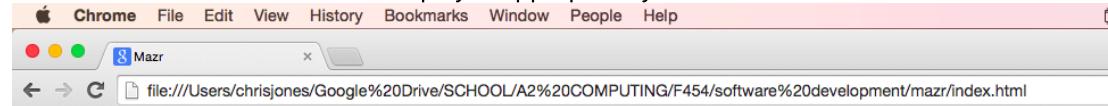
```
113      <div class="end-of-level-menu page">
114          <div class="buttons">
115              <h2>Congratulations!</h2>
116              <p>You completed the maze in <span id="time"></span></p>
117              <p>You took <span id="steps"></span> steps to complete the
118                  maze.</p>
119              <p>Your progress has been saved.</p>
120              <button class="btn-next-level">Next level</button>
121              <button class="btn-quit">Quit to main menu</button>
122      </div>
123  </div>
```

Finally, just before the closing body tag, all the necessary scripts must be linked to so that they are included when a user's browser requests the webpage. Each of the files in the scripts folder is included as the source of its own script tag and included in the order in which the dependencies lie (for example, sprite.js must be included before SpriteBatch.js because the Sprite objects will be required by a SpriteBatch object). This is shown in the image below.

```
131      <!-- Include all the necessary scripts -->
132      <script src="scripts/camera.js" type="text/javascript"></script>
133      <script src="scripts/sprite.js" type="text/javascript"></script>
134      <script src="scripts/SpriteBatch.js" type="text/javascript"></script>
135      <script src="scripts/timer.js" type="text/javascript"></script>
136      <script src="scripts/player.js" type="text/javascript"></script>
137      <script src="scripts/ExitTile.js" type="text/javascript"></script>
138      <script src="scripts/maze.js" type="text/javascript"></script>
139      <script src="scripts/engine.js" type="text/javascript"></script>
140  </body>
141  </html>
```

### Alpha testing the HTML

With the HTML file completed, I opened the webpage in the Google Chrome web browser in order to review that all of the content was displayed appropriately.



## Mazr

The endless maze solving puzzle game



## Help

You are the coloured square.

You are placed in a randomly generated maze. All mazes are solvable. Move onto the orange square to complete the maze.



## Help

You can move around the maze with the arrow keys, providing no walls block your path. You will leave a coloured trail.



After checking each element against the corresponding HTML I'd written I was satisfied all necessary content was present. Note that the image links are broken because the images that will appear on the help menus haven't been made yet (there'll be made once the system has been developed) and as such this is an unavoidable issue at this moment in time. Regardless, the webpage displays correctly as this particular moment.

## CSS

This section pertains to the development of Output Requirements 19, 20, 21, 22, 23, 24 and 25 of the Requirements Specification.

Now, styling can be applied in the form of stylesheet.css to ensure that it appears aesthetically as it does in the Nature of The Solution's screen mock-ups and layouts. The “Styling the user interface with CSS” section in the Nature of The Solution will be used to define the selectors that make up the CSS file. First, html and the \* selector are used to set the box sizing to a border-box layout. Border-box is a layout system that helps to preserve widths set out before content is added to elements. This will allow the menus to retain their sizing correctly. The noscript selector is used to style the error message that is contained within it, thus the red colour attribute and 28-pixel size font. The body is also given some styling options.



```
stylesheet.css
1 html {
2     box-sizing: border-box;
3 }
4 *, *:before, *:after {
5     box-sizing: inherit;
6 }
7
8 ▼ noscript {
9     font-size: 28px;
10    font-weight: normal;
11    font-style: italic;
12    color: red;
13    width: 512px;
14    display: block;
15    margin: auto;
16 }
17
18 ▼ body {
19     background: #ddd;
20     color: #222;
21
22     font-family: Helvetica;
23     font-weight: bold;
24     text-align: center;
25
26     margin-left: auto;
27     margin-right: auto;
28 }
```

Next, the text styling is applied. This includes the hgroup, h2, paragraph and “a” link selectors. The under-text class is also included, which will style the small caption in the continue button of index.html. This is shown in the image below.

```

30  hgroup h1 {
31      font-size: 100px;
32  }
33  hgroup p {
34      font-size: 16px;
35  }
36
37  hgroup, footer {
38      padding: 12px 0px;
39  }
40
41  h2 {
42      font-size: 30px;
43  }
44
45  p {
46      padding: 0px 50px;
47  }
48
49  a {
50      color: #222222;
51      font-style: italic;
52      font-weight: lighter;
53  }
54
55  .under-text {
56      font-size: 12px;
57  }

```

Next the canvas and page layout styles are applied, as well as the containers and the arrow button class selector. This is necessary in order to make sure all the menus are styled and sized similarly. These selectors are shown in the image below.

```

59  canvas {
60      border: 4px solid #222;
61  }
62
63  .page {
64      background: #333;
65      display: inline-flex;
66      border: 4px solid #222;
67
68      color: #ddd;
69
70      width: 520px;
71      height: 520px;
72
73      margin: auto;
74  }
75  canvas, .page {
76      padding: 0;
77      margin-bottom: 0;
78  }
79
80  .buttons {
81      margin: auto;
82  }
83
84  .btn-arrow {
85      display: inline;
86      width: 150px;
87      border: 5px solid #333333;
88      margin: 0;
89  }

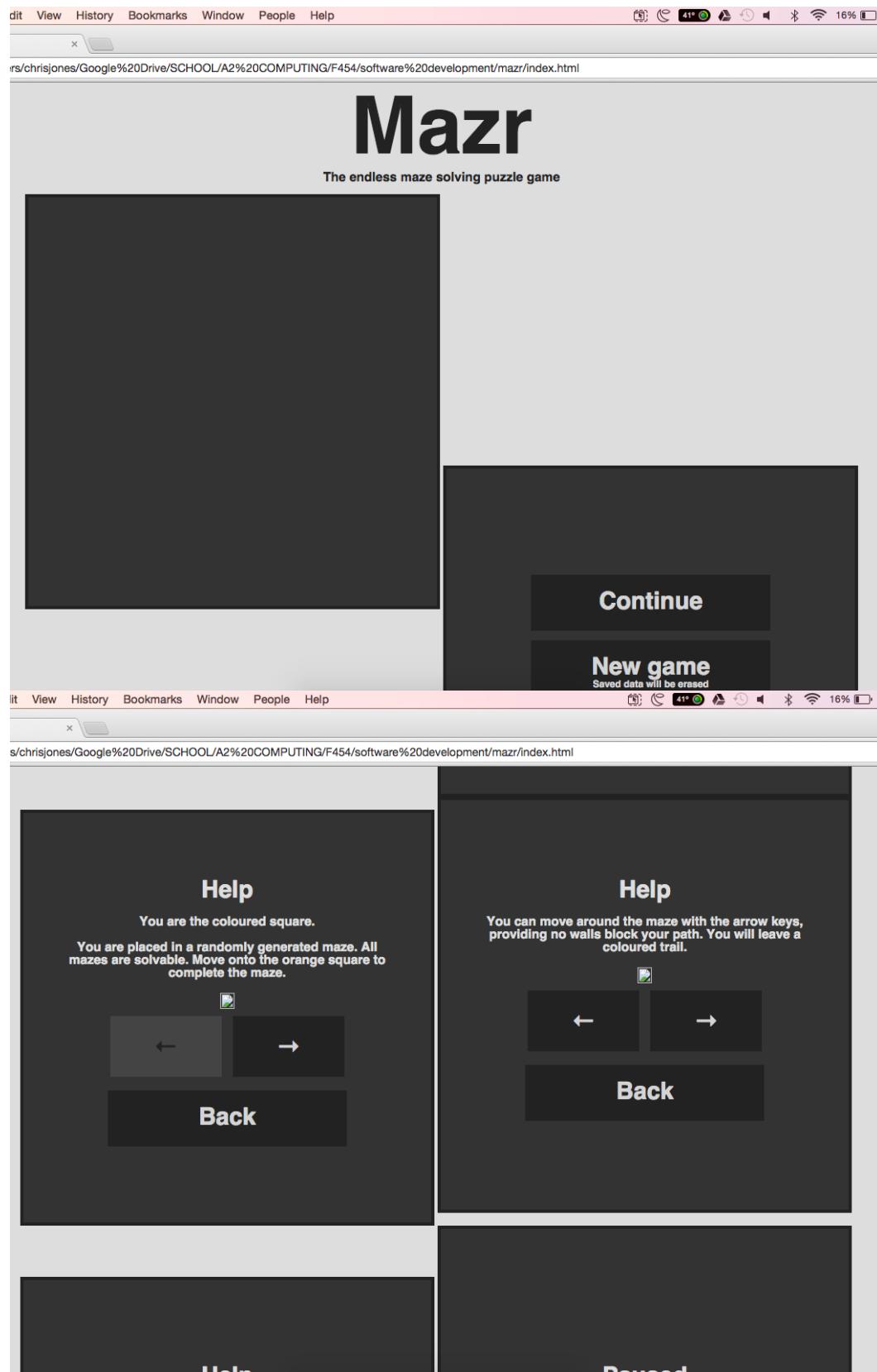
```

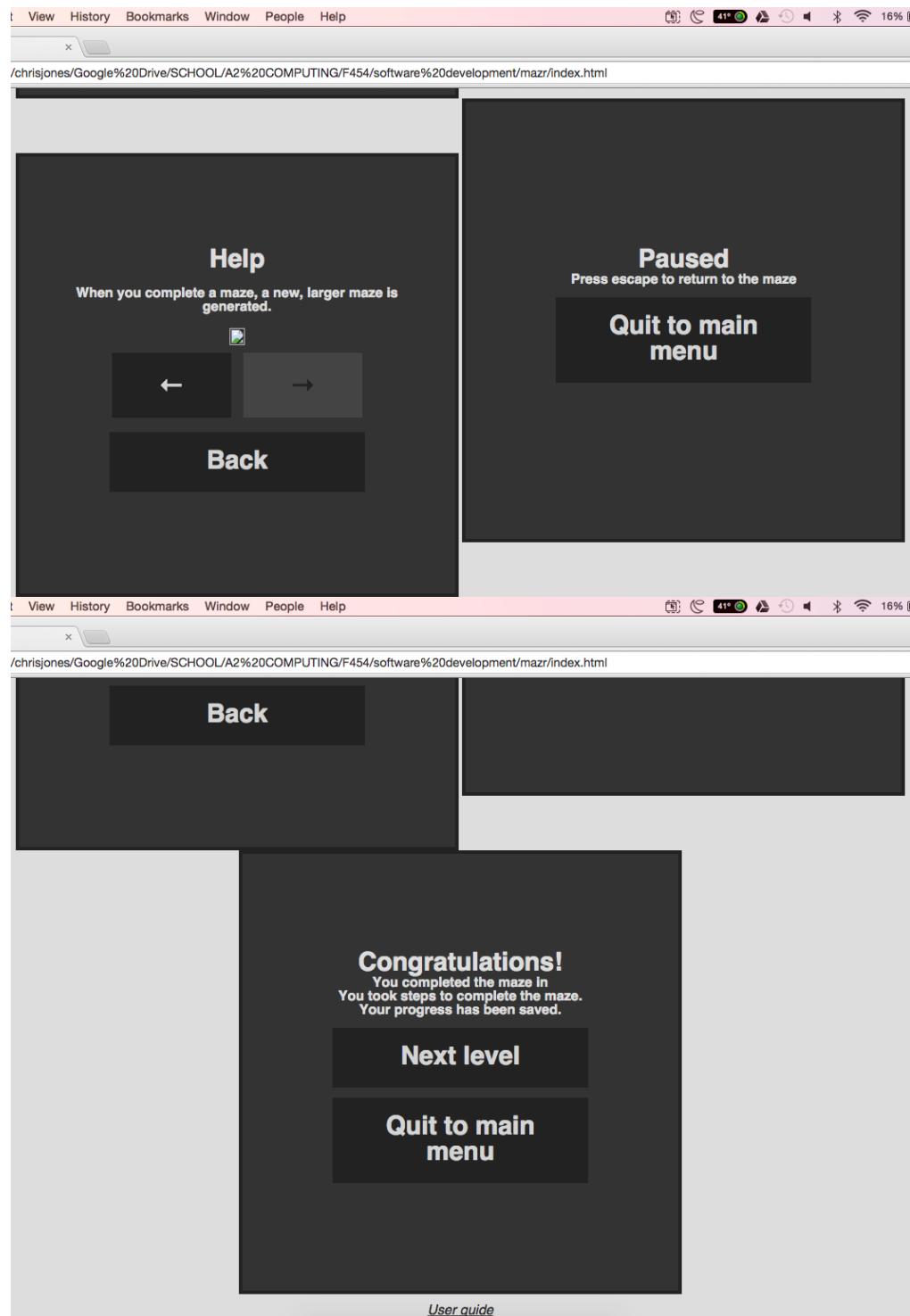
All of the states associated with the buttons are added to the stylesheet next. This is the most complex amount of styling required and entered into the button element selector below. These are separated into the default button element: the active state (when the button is pressed down this style is applied), the focus state (when the element is selected) and disabled (when the element's control is disabled). This is shown in the image below.

```
90 button {
91   display: block;
92   margin: 12px auto;
93   padding: 8px 12px;
94
95   width: 300px;
96
97   background: #222;
98   border: 0px;
99
100  padding: 20px;
101
102  /* Text */
103  font-size: 30px;
104  font-family: Helvetica;
105  font-weight: bold;
106  color: #d4d4d4;
107 }
108 button:active {
109   outline: none;
110
111   -webkit-box-shadow: inset 0px 4px 20px 4px rgba(0,0,0, 0.4);
112   -moz-box-shadow: inset 0px 4px 20px 4px rgba(0,0,0, 0.4);
113   box-shadow: inset 0px 4px 20px 4px rgba(0,0,0, 0.4);
114 }
115 button:focus {
116   outline: none;
117
118   color: #222;
119   background: hsl(145, 80%, 40%);
120
121
122 }
123 button:disabled {
124   outline: none;
125
126   background: #444;
127   color: #222;
128 }
129
130
```

### Alpha testing the CSS

The stylesheet.css file has now been completed. Below are images of the browser window showing the styled webpage. Note that all of the menus are being displayed at once - as no JavaScript is present that is hiding the correct set of menus. This will be rectified and correct functionality added later.





Due to development purposes, the styling will be disabled until the end of the development of engine.js. This is because for alpha testing styling will simply make evaluating the result of scripts more difficult until the jQuery event handling code has been added. Therefore, the stylesheets are commented out. This is shown in the image below.

```

8      <!-- Link to stylesheets -->
9      <!--<link rel="stylesheet" href="styles/reset.css" />
10     <link rel="stylesheet" href="styles/stylesheet.css" />-->
11

```

With the styling removed, JavaScript can now start being written for the development of Mazr.

**Camera class function creation**

This section pertains to the development of the foundations of Processing Requirement 18 and Output Requirements 33 and 34.

In order to instantiate a camera object, we first need a class definition. However, JavaScript has no native ability to create a class so a function is used to define an object's structure, attributes and methods. Note that this pattern is repetitive within the whole project – most class definitions will be created with functions.

```
camera.js
/*
  camera.js
  Camera object
  _____
  Written on the 29/10/2014 by Chris Jones.

  This file represents the Camera data structure represented in the Data
  Structures section of the design specification.

  The function Camera represents a function from which a Camera object can be
  instantiated. This will represent the Camera data structure.

  This class is used by the Maze object to keep track of the current position
  of the player's view at any given time. It also allows the focus of the maze
  to be adjusted and not fixed at a given position.

  The camera has 4 attributes:
    - x: The pixel coordinate of the camera's position in the x axis.
    - y: The pixel coordinate of the camera's position in the y axis.
    - width: The width of the camera's viewport in pixels
    - height: The height of the camera's viewport in pixels.
*/
function Camera(x, y, width, height) {
  /* Insert attributes here */
}
```

Firstly, in the camera.js file, a multiline comment header was created. This contains some information about the file and its interrelationship with the rest of the software. I also created an empty function that takes four parameters – x, y, width and height. These will be used to populate the attributes of the function later on.

Next, the x attribute is defined. Code included in order to ensure that the validation requirements detailed in the data structures section of the nature of the solution are met. A set of “if, else-if, else” statements is used to determine whether the parameter x is of the primitive data-type Number, in which case its value is assigned to the attribute x (this.x). If the parameter is null or undefined, then it is assigned a default value of zero. Otherwise, if neither condition is met, an error is thrown by the system. This code is shown in an image below.

```

20           width: The width of the camera's viewport in pixels.
21     */
22   function Camera(x, y, width, height) {
23
24     // Check if x supplied is of primitive type Number
25     if (typeof x === "number") {
26       // Assign value of parameter x to property x
27       this.x = x;
28     }
29     // Check if no parameter was passed (undefined) or a null parameter was passed
30     else if (x === undefined || x === null) {
31       // Assign x property default value of 0
32       this.x = 0;
33     }
34     else {
35       // Throw an error if above conditions are not met
36       // Note errors will be caught in the main game loop.
37       throw "Argument error. x coordinate must a number.";
38     }
39   }
40 }
41
42

```

This code is almost formulaically repeated for the y parameter, except of course with the replacement of the attribute y within the selection statement. Again, assigns the attribute y (this.y) the value of y if it is a primitive data-type Number, or it defaults to zero if y is null or undefined, or throws an error if neither condition is met.

```

41   // Check if y supplied is of primitive type Number
42   if (typeof y === "number") {
43     // Assign value of parameter y to property y
44     this.y = y;
45   }
46   // Check if no parameter was passed (undefined) or a null parameter was passed
47   else if (y === undefined || y === null) {
48     // Assign y property default value of 0
49     this.y = 0;
50   }
51   else {
52     // Throw an error if above conditions are not met.
53     // Note errors will be caught in the main game loop.
54     throw "Argument error. y coordinate must a number."
55   }
56
57 }
58

```

Now the width attribute is defined. This is again enclosed in the same set of if, else-if and else statements as x and y. However, the default property of width is 512 instead of zero.

```

56   // Check if the width supplied is of primitive type Number
57   if (typeof width === "number") {
58     // Assign value of parameter width to property width
59     this.width = width;
60   }
61   // Check if no parameter was passed (undefined) or a null parameter was passed
62   else if (width === undefined || width === null) {
63     // Assign width property default value of 512
64     this.width = 512;
65   }
66   else {
67     // Throw an error if above conditions are not met.
68     // Note errors will be caught in the main game loop.
69     throw "Argument error. Width must a number."
70   }
71
72 }
73

```

Finally, the height attribute is defined. This is almost identical to the width attribute definition, except the replacement of “width” identifiers with “height” ones.

```

73  // Check if the height supplied is of primitive type Number
74  if (typeof height === "number") {
75      // Assign value of parameter height to property he
76      this.height = height;
77  }
78  // Check if no parameter was passed (undefined) or a null parameter was passed
79  else if (height === undefined || height === null) {
80      // Assign height property default value of 512
81      this.height = 512;
82  }
83  else {
84      // Throw an error if above conditions are not met.
85      // Note errors will be caught in the main game loop.
86      throw "Argument error. Height must a number."
87  }
88
89 }
90

```

### Camera class function alpha testing

Now that the camera function has been completed it is time to perform some initial alpha testing. This will be done using the JavaScript console that comes include with the Google Chrome web browser.

Most of the alpha testing in the software development section will involve either entering small amounts of code into the console and observing its output or writing additional test code into the JavaScript files of the game and observe their output in a browser window. Note the inclusion of any test code will be indicated by comments within the code and not be included in the final product.

Below is a screenshot of the JavaScript console. The line in light blue was entered manually into the console in order to define a new Camera object 'c' with pixel coordinates (16,16) and width and height dimensions of 256. Observation of the return value of this statement shows that the properties are correctly assigned.

```

< undefined
> c = new Camera(16, 16, 256, 256)
< Camera {x: 16, y: 16, width: 256, height: 256}
>

```

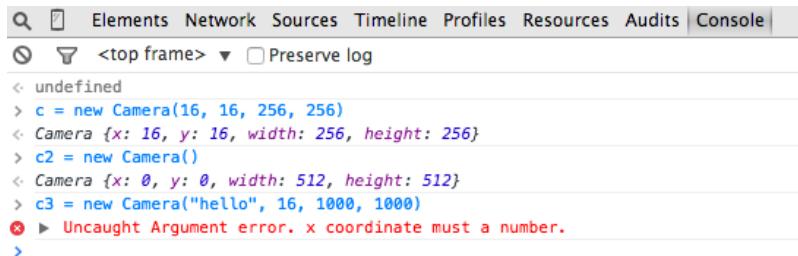
Next, observe the second light blue coloured line of input into the console. The camera object 'c2' is defined with no parameters passed. This ensures that the default values are used (x = 0, y = 0, width = 512, height = 512). Note from the second line of output that all of the properties have the correct default values to which they should have assigned.

```

< undefined
> c = new Camera(16, 16, 256, 256)
< Camera {x: 16, y: 16, width: 256, height: 256}
> c2 = new Camera()
< Camera {x: 0, y: 0, width: 512, height: 512}
>

```

Finally, we need to check that the error throwing final condition works. A camera object 'c3' was created with the first parameter being passed as a string. As this is neither primitive type Number, null, or undefined then the error should be thrown. Note in the red text that the error that was written specifically on line 38 of the camera.js file is output.



The screenshot shows the 'Console' tab of a browser's developer tools. It displays the following JavaScript code and its execution results:

```
< undefined
> c = new Camera(16, 16, 256, 256)
< Camera {x: 16, y: 16, width: 256, height: 256}
> c2 = new Camera()
< Camera {x: 0, y: 0, width: 512, height: 512}
> c3 = new Camera("hello", 16, 1000, 1000)
✖ > Uncaught Argument error. x coordinate must a number.
```

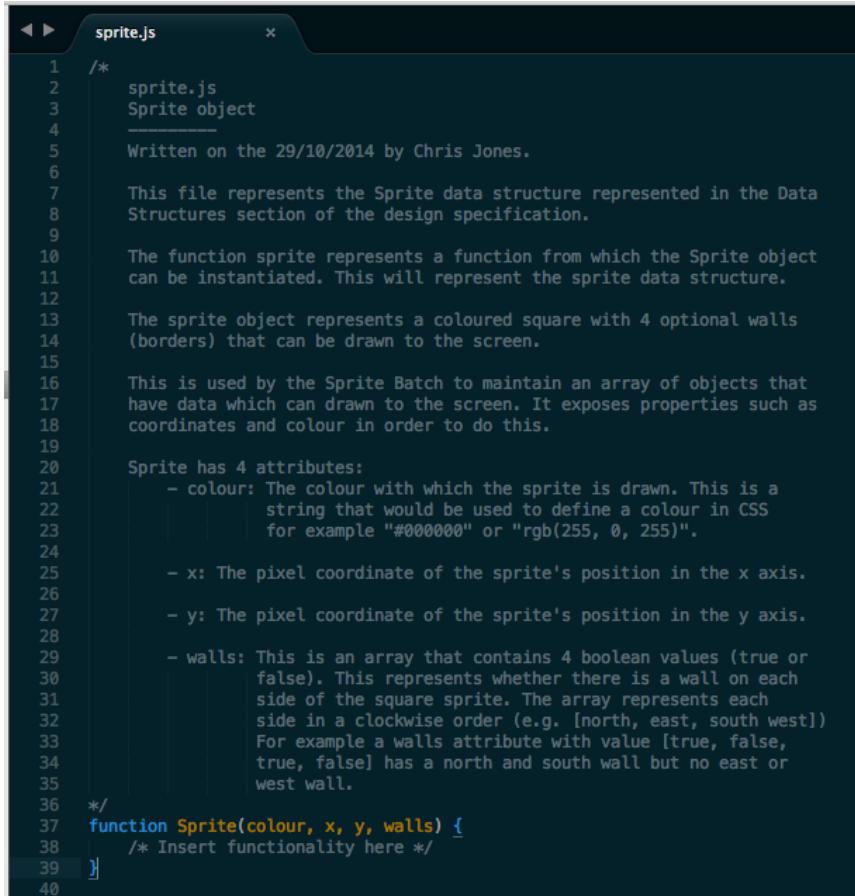
This is satisfactory alpha testing to show that the camera class functions correctly.

### Sprite class function

This section pertains to the development of the foundations of Output Requirements 31 and 32. Note also that this section provides the foundations for almost all of the Output Requirements that require drawing to the canvas, although do not explicitly mention it. This Sprite Class forms the foundation of drawing for the whole system.

Working sequentially through the data structures in the nature of the solution, the next data structure, which must be implemented in JavaScript code, is the Sprite. A sprite represents a rectangle with borders that can be drawn in the game world.

In the sprite.js file, a multiline comment containing a header is created which contains some important information about the function. An empty function with identifier sprite and four properties is also included with parameters colour, x, y and walls.



```

1  /*
2   *      sprite.js
3   *      Sprite object
4   *
5   *      Written on the 29/10/2014 by Chris Jones.
6   *
7   *      This file represents the Sprite data structure represented in the Data
8   *      Structures section of the design specification.
9   *
10  *      The function sprite represents a function from which the Sprite object
11  *      can be instantiated. This will represent the sprite data structure.
12  *
13  *      The sprite object represents a coloured square with 4 optional walls
14  *      (borders) that can be drawn to the screen.
15  *
16  *      This is used by the Sprite Batch to maintain an array of objects that
17  *      have data which can drawn to the screen. It exposes properties such as
18  *      coordinates and colour in order to do this.
19  *
20  *      Sprite has 4 attributes:
21  *          - colour: The colour with which the sprite is drawn. This is a
22  *                      string that would be used to define a colour in CSS
23  *                      for example "#000000" or "rgb(255, 0, 255)".
24  *
25  *          - x: The pixel coordinate of the sprite's position in the x axis.
26  *
27  *          - y: The pixel coordinate of the sprite's position in the y axis.
28  *
29  *          - walls: This is an array that contains 4 boolean values (true or
30  *                      false). This represents whether there is a wall on each
31  *                      side of the square sprite. The array represents each
32  *                      side in a clockwise order (e.g. [north, east, south west])
33  *                      For example a walls attribute with value [true, false,
34  *                      true, false] has a north and south wall but no east or
35  *                      west wall.
36  */
37  function Sprite(colour, x, y, walls) {
38      /* Insert functionality here */
39  }
40

```

In exactly the same fashion as the Camera function, the x and y attributes are assigned first. If, else-if and else statements are again used to ensure the parameters are entered correctly otherwise default values are assigned or an error is thrown. This is exactly the same as the Camera class and is shown in the image below.

```

37▼ function Sprite(colour, x, y, walls) {
38
39    // Check if x supplied is of primitive type Number
40▼    if (typeof x === "number") {
41        // Assign value of parameter x to attribute x
42        this.x = x;
43    }
44    // Check if no parameter was passed (undefined) or a null parameter was passed
45▼    else if (x === undefined || x === null) {
46        // Assign x attribute default value of 0
47        this.x = 0;
48    }
49▼    else {
50        // Throw an error if above conditions are not met
51        // Note errors will be caught in the main game loop.
52        throw "Argument error. x coordinate must a number.";
53    }
54
55    // Check if y supplied is of primitive type Number
56▼    if (typeof y === "number") {
57        // Assign value of parameter y to attribute y
58        this.y = y;
59    }
60    // Check if no parameter was passed (undefined) or a null parameter was passed
61▼    else if (y === undefined || y === null) {
62        // Assign x attribute default value of 0
63        this.y = 0;
64    }
65▼    else {
66        // Throw an error if above conditions are not met.
67        // Note errors will be caught in the main game loop.
68        throw "Argument error. y coordinate must a number."
69    }
70▼ }
71

```

Next, the colour attribute is defined. There is an initial if statement to check if it is of a type string – in this case the attribute colour is assigned the parameter colour. Otherwise if the parameter colour is undefined or null it is assigned the string value “#000000”, this is a CSS Hexadecimal definition of the colour black. If neither of these conditions is met then an error is thrown. Below is an image that shows this code added into the file.

```

// Set the colour of the sprite
if (typeof colour === "string") {
    // Assign value of parameter colour to attribute colour
    this.colour = colour;
}
// Check if no parameter was passed (undefined) or a null parameter was passed
else if (colour === undefined || colour === null) {
    // Assign colour attribute default value of "#000000" (black)
    this.colour = "#000000";
}
else {
    // Throw an error if above conditions are not met.
    // Note errors will be caught in the main game loop.
    throw "Argument error. colour must a string."
}

```

Next the walls attribute must be defined. This is done in the same way as the other attributes – by containing three conditions set by if, else-if, and else statements. The first if statements checks if the walls parameter is an instance of the Array object. Note that we no longer use the ‘typeof’ keyword as this only works for primitive types. For objects, such as arrays, the ‘instanceof’ keyword must be used. If it is then the parameter walls is assigned to the attribute walls. If ‘walls’ is null or

undefined it is assigned an array of 4 false Boolean values, representing no walls. If neither of these conditions is met an error is thrown. Below is an image showing the code added to the file.

```

86     // Set the value of the attribute walls
87     if (walls instanceof Array && walls.length == 4) {
88         this.walls = walls;
89     }
90     // Check if no parameter was passed (undefined) or a null parameter was passed
91     else if (walls === undefined || walls === null) {
92         // Assign walls attribute default value of [false, false, false, false]
93         this.walls = [false, false, false, false];
94     }
95     else {
96         // Throw an error if above conditions are not met.
97         // Note errors will be caught in the main game loop.
98         throw "Argument error. walls must be an array with 4 boolean elements"
99     }
100 }
101 }
102
103

```

With this code added it is now time to alpha test the class using the JavaScript console.

### Alpha testing the Sprite function

The first item of alpha testing to begin with is to instantiate a new Sprite with all of the arguments obeying validation rules, as shown in the light blue input to the console. This produces an output that can be observed as correct – all of the attributes are assigned correctly.

```

Elements Network Sources Timeline Profiles Resources Audits | Console
<top frame> ▾ □ Preserve log
> s = new Sprite("blue", 0, 50, [false, false, false, true])
<- ▾ Sprite {x: 0, y: 50, colour: "blue", walls: Array[4]} ⓘ
  colour: "blue"
  ▾ walls: Array[4]
    0: false
    1: false
    2: false
    3: true
    length: 4
    ▶ __proto__: Array[0]
  x: 0
  y: 50
  ▶ __proto__: Sprite

```

Next we instantiate a sprite object ‘s2’ which has no arguments passed to it. This is to test that it defaults to the correct value. This can be seen in the light blue text without the output from the console below it. The output clearly shows all of the properties are defined correctly with the correct default values.

```

> s2 = new Sprite()
<- ▾ Sprite {x: 0, y: 0, colour: "#000000", walls: Array[4]} ⓘ
  colour: "#000000"
  ▾ walls: Array[4]
    0: false
    1: false
    2: false
    3: false
    length: 4
    ▶ __proto__: Array[0]
  x: 0
  y: 0
  ▶ __proto__: Sprite

```

Next, the object ‘s3’ is instantiated with one incorrect parameter, y, being supplied as a string rather than a number. This produces the correct error as shown below.

```
> s3 = new Sprite("blue", 100, "green", [false,false,false,false])
✖ ▶ Uncaught Argument error. y coordinate must a number.
```

Sprite object 's3' is instantiated with the colour argument passed as a Number and not as a string. This produces the correct error shown below, informing the console user that the colour argument must be passed as a string.

```
> s4 = new Sprite(100, 100, 100, [false,false,false,false])
✖ ▶ Uncaught Argument error. colour must a string.
```

In this alpha test, a Sprite 's5' is instantiated. This is passed correct arguments, except that the last element of the Boolean array 'walls' is a string rather than a Boolean. Rather incorrectly, by observing the output, no error is returned and in fact this is stored in the attribute walls. This is highlighted in the output below.

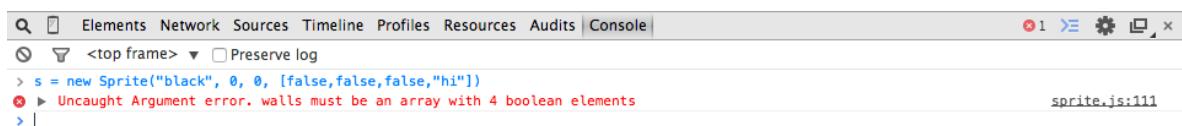
```
> s5 = new Sprite("black", 0, 0, [false, false, false, "hi"])
< ▼ Sprite {x: 0, y: 0, colour: "black", walls: Array[4]} ⓘ
  colour: "black"
  ▼ walls: Array[4]
    0: false
    1: false
    2: false
    3: "hi" highlighted
    length: 4
  ▶ __proto__: Array[0]
  x: 0
  y: 0
  ▶ __proto__: Sprite
```

To remedy this issue, some additional code must be written that can check that that the data-type of each element in the array of walls is a Boolean value. This will be done by assuming that they are all correct by defining the variable correctType as true, iterating through each of the four elements, then setting correctType false if any of the elements is not a Boolean value. Below is an image of the amended code shown inserted into the if-statement. The amended code is squared off in red.

```

87      // Set the value of the attribute walls
88  ▼  if (walls instanceof Array && walls.length == 4) {
89      // Declare local variable correctType as true
90      // Assume at this point the elements of walls are all boolean data-types
91      var correctType = true;
92
93      // Iterate through each element of walls with index i
94  ▼  for (var i = 0; i < walls.length; i++) {
95          // Check whether the type is NOT boolean
96  ▼  if (typeof walls[i] != "boolean") {
97              // Set correctType to false if not a boolean
98              correctType = false;
99              // Break from the loop if a non-boolean type is detected
100             break;
101         }
102     }
103
104     // If correctType is true (when all 4 elements are boolean)
105  ▼  if (correctType)
106         // Assign parameter walls to attribute walls
107         this.walls = walls;
108  ▼  else
109         // Throw an error at this point if not all data types are boolean
110         // Note errors will be caught in the main game loop.
111         throw "Argument error. walls must be an array with 4 boolean elements"
112     }
113
114  ▼  // Check if no parameter was passed (undefined) or a null parameter was passed
115  ▼  else if (walls === undefined || walls === null) {
116      // Assign walls attribute default value of [false, false, false, false]
117      this.walls = [false, false, false, false];
118  ▼  } else {
119      // Throw an error if above conditions are not met.
120      // Note errors will be caught in the main game loop.
121      throw "Argument error. walls must be an array with 4 boolean elements"
122  }
123 }
```

With this new code added, it is time to repeat the alpha testing by opening the JavaScript console in a Google Chrome and entering a line of input; instantiating Sprite s and passing it a string in the final argument walls. The second line output from the console shows the error has been thrown. Paying attention to the line of the error shows its on line 111 of the sprite.js file – where the new error is inserted.

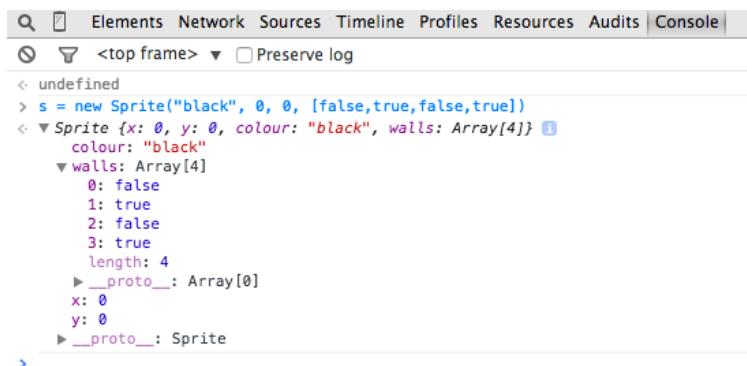


The screenshot shows the Google Chrome Developer Tools Console tab. The URL bar at the top shows 'Console'. The console log area contains the following entries:

```

Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> ▾ □ Preserve log
> s = new Sprite("black", 0, 0, [false,false,false,"hi"])
✖ ▶ Uncaught Argument error. walls must be an array with 4 boolean elements
sprite.js:111
> |
```

Finally, one more test is made to ensure that supplying the final argument with an array of four Boolean elements works correctly. This is shown working correctly in the console screenshot below.



The screenshot shows the Google Chrome Developer Tools Console tab. The URL bar at the top shows 'Console'. The console log area contains the following entries:

```

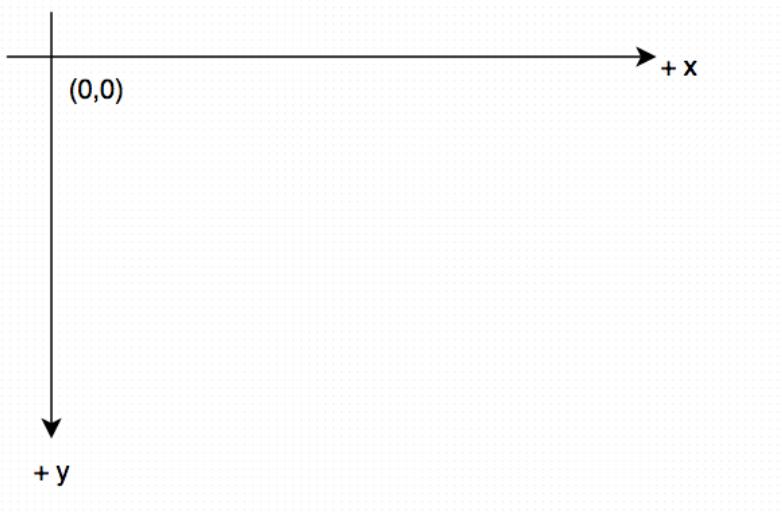
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> ▾ □ Preserve log
<- undefined
> s = new Sprite("black", 0, 0, [false,true,false,true])
<- ▾ Sprite {x: 0, y: 0, colour: "black", walls: Array[4]} ⓘ
  colour: "black"
  ▾ walls: Array[4]
    0: false
    1: true
    2: false
    3: true
    length: 4
    ▶ __proto__: Array[0]
  x: 0
  y: 0
  ▶ __proto__: Sprite
> |
```

**SpriteBatch function creation**

This section pertains to the development of the foundations of Output Requirements 31 and 32. Note also that this section provides the foundations for almost all of the Output Requirements that require drawing to the canvas, although do not explicitly mention it.

A SpriteBatch in the context of programming Mazr contains a batch of sprites that are drawn to the canvas. It is the role of the SpriteBatch to use the attributes of each sprite and a camera to determine the styling and positioning of all the sprites.

It is important to note that the coordinate axes of the canvas differ from how they'd be imagined in a conventional mathematical setting. Below in a diagram represents the coordinate axes of a typical canvas. The origin, or point (0,0), is shown on the diagram and represents the top-left corner of a canvas HTML element. From this point the x-axis increases to the right and the y-axis increases downwards. This is possibly counterintuitive; to move a sprite downwards you must increase its y value.



To begin with the development of the SpriteBatch, SpriteBatch.js is opened in Sublime Text and a multiline comment header is added containing some details about the function. Again, this function will be used to instantiate objects in later code (in this case the Maze function). Then an empty function definition is added with the two necessary parameters from data structures, 'camera' and 'canvasContext'.

```

1  /*
2   *   SpriteBatch.js
3   *   SpriteBatch object
4   *
5   *   Written on the 30/10/2014 by Chris Jones
6   *
7   *   This file represents the SpriteBatch data structure represented in the Data
8   *   Structures section of the design specification.
9   *
10  *   The function SpriteBatch represents a function from which the SpriteBatch
11  *   object can be instantiated. This will represent the spriteBatch data
12  *   structure.
13  *
14  *   SpriteBatch controls the drawing of an array of sprites to a canvas through
15  *   a canvas' drawing context. It maintains an array of sprites that are drawn
16  *   one by one to the canvas in their correct position by drawing coloured
17  *   rectangles. Once all of the rectangles are drawn then paths are created for
18  *   each of the sprite's walls. This is encapsulated in a draw function
19  *
20  *   SpriteBatch has 3 attributes:
21  *       - batch: An array, declared empty upon instantiation, that contains
22  *           the sprites that are to be drawn to the canvas. Sprites are
23  *           added in order relative to their z-index on the canvas. For
24  *           example, adding two sprites with exactly the same positions
25  *           will cause the last sprite added to be drawn on top of the
26  *           first sprite added.
27  *
28  *       - canvasContext: A reference to the 2D drawing context of the canvas
29  *           onto which the sprites will be drawn.
30  *
31  *       - camera: Reference to the camera object with which the game is being
32  *           viewed.
33  *
34  *   SpriteBatch has one method:
35  *       - draw: This draws all of the sprites in batch to canvas serially
36  *           (one after another). All of the sprite's backgrounds are
37  *           first drawn and then the walls are all drawn together on
38  *           top. This ensures no sprites are drawn on top of the walls.
39  */
40  function SpriteBatch(canvasContext, camera) {
41      /* ... Functionality here ... */
42 }

```

The batch attribute is then defined and has an empty array literal assigned to it.

```

44  function SpriteBatch(canvasContext, camera) {
45      // Contains a batch of sprites to draw
46      // Initialise as an empty array
47      // As this is an attribute of the SpriteBatch object
48      // it can be populated later
49      this.batch = [];
50  }
51

```

Next, the attribute canvasContext is assigned to. In the same style as the Camera and Sprite class functions, this will be contained with an if-statement in order to validate the parameters. However, there is no need to assign it a default value as there is no default for a canvas-drawing context that has been extracted from the Document Object Model of the HTML page. This is shown inserted into the class function in the image below.

```

44     function SpriteBatch(canvasContext, camera) {
45         // Contains a batch of sprites to draw
46         // Initialise as an empty array
47         // As this is an attribute of the SpriteBatch object
48         // it can be populated later
49         this.batch = [];
50
51         // Reference to canvasContext on which the batch is drawn
52         // Note there is no defaulting code path for the canvasContext
53         // This is because it would be meaningless for a spritebatch
54         // to have a canvas drawing context defined itself
55         if (canvasContext === undefined || canvasContext === null) {
56             throw "Argument error. Improper canvas context specified.";
57         } else {
58             // Assign canvasContext parameter to canvasContext attribute
59             this.canvasContext = canvasContext;
60         }
61
62     }
63 }
```

In the same format the camera attribute is defined, with the bottom of the class function with this addition shown below.

```

62         // Reference to camera to which the sprites will be transformed
63         // Note there is no defaulting code path for the camera
64         // This is because it would be meaningless for a spritebatch
65         // to draw to its own camera
66         if (camera === undefined || camera === null) {
67             throw "Argument error. Improper camera specified.";
68         } else {
69             // Assign camera parameter to camera attribute
70             this.camera = camera;
71         }
72     }
73 }
```

Now it was time to define the draw function. First the functionality that draws the rectangles of each sprite to the canvas was written. A for loop is used to iterate through each of the sprites with an index of i. Then, an 'if statement' is used to determine if the sprite in the batch array with index i is within the view of the camera, otherwise it does not bother drawing it. Then, the fillStyle property of the canvasContext reference attribute is set to equal the colour property of the sprite in the batch array with index i. Finally, the fillRect method of canvasContext is called and passed four arguments, the x coordinate, the y coordinate, the width and the height. The global constant TILE\_SIZE is used for the width and height arguments and the x and y coordinates, offset by the camera, are used as the x and y parameters. This is shown at the end of the SpriteBatch class function in the image below.

```

73     // draw function acts as method that handles the drawing of sprites in batch
74     this.draw = function() {
75         // Iterate through each sprite in the batch
76         for (var i in this.batch) {
77             // Check that the i'th sprite is located
78             // within the bounds of the camera
79             // This means computer resources aren't wasted rendering
80             // a sprite that isn't going to be seen
81
82             // Subtracting the camera's coordinate's from those of
83             // the sprite effectively moves the sprites with respect
84             // to a stationary camera. This has the effect of the camera
85             // moving and the sprites remaining stationary
86             if ((this.batch[i].x - this.camera.x < this.camera.width) &&
87                 (this.batch[i].y - this.camera.y < this.camera.height)) {
88
89                 // Set the fillStyle of the canvas equal
90                 // to the colour of the sprite
91                 this.canvasContext.fillStyle = this.batch[i].colour;
92
93                 // Draw a rectangle onto the canvas
94                 // At the x and y coods of the sprite minus the camera's coods
95                 // The size of the rectangle should equal
96                 // the TILE_SIZE constraint
97                 this.canvasContext.fillRect(
98                     this.batch[i].x - this.camera.x,
99                     this.batch[i].y - this.camera.y,
100                    TILE_SIZE,
101                    TILE_SIZE
102                );
103            }
104        }
105    }
106 }

```

### Initial Alpha testing of the SpriteBatch

Now was a good time to check that the rectangle drawing functionality of draw behaved as intended. Therefore, some test code was appended to the end of the file. This is seen below.

```

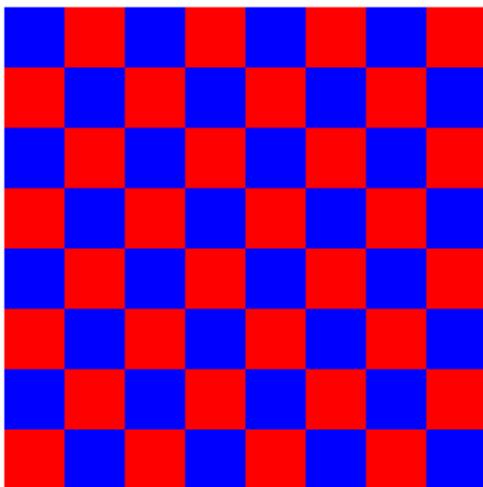
108 // Initialise a temporary constant for tile size
109 const TILE_SIZE = 32;
110
112 // Get the canvas drawing context
113 var canvas = document.getElementById('canvas');
114
115 canvas.width = 512;
116 canvas.height = 512;
117
118 var canvasContext = canvas.getContext('2d');
119
120 // Instantiate a new SpriteBatch
121 var spriteBatch = new SpriteBatch(canvasContext, new Camera());
122
123 var alternate = false;
124
125 // Fill it with a grid of 8 by 8 sprites
126 for (var i = 0; i < 8; i++) {
127     for (var j = 0; j < 8; j++) {
128         // Push a new sprite to the batch
129         spriteBatch.batch.push(
130             new Sprite(
131                 alternate ? "red" : "blue",
132                 i*TILE_SIZE,
133                 j*TILE_SIZE,
134                 [false, false, false, false]
135             )
136         );
137         // Invert alternate
138         alternate = !alternate;
139     }
140     // Invert alternate
141     alternate = !alternate;
142 }
143
144 // Draw the sprites
145 spriteBatch.draw();
146

```

This code from lines 109 to 118 contains some initial boilerplate functionality that is required to access drawing canvas functionality. Then, a new SpriteBatch is defined with identifier SpriteBatch on line 121. A double-nested for loop is used to generate a grid of sprites, each of which is pushed

to the batch of spriteBatch. The Boolean variable alternate is used in conjunction with the ternary operating on line 131 to alternate the colour of each square in order to produce a checkerboard pattern. This is the pattern we'd expect if we visited the webpage. Within the nest of both loops alternate is inverted so that it value changes with each change to the counter variables i and j.

Below is an image of the webpage with the output of the canvas. Note this is exactly what was expected from the additional test code that was written. The test code will remain in the file so that when the rest of the draw function is completed it can still be used to test whether the walls are correctly rendered on the canvas.



### Completion of draw method

*This section pertains to the development of the foundations of Output Requirements 31 and 32. Note also that this section provides the foundations for almost all of the Output Requirements that require drawing to the canvas, although do not explicitly mention it.*

The following lines were added to the end of the draw method. They involve casting values to some of the properties of the canvasContext reference attribute. The first sets the colour of the strokeStyle property (the colour in which lines are drawn to the canvas) equal to the WALL\_COLOUR constant (note we haven't defined this yet and is defined later in the engine.js file, although for purposes of test will be defined temporarily). The lineWidth property is set to one sixteenth of the TILE\_SIZE constant. Next, the beginPath method of canvasContext is called – this has the effect of storing the paths that we are draw ready for output to the canvas. Again, in the same fashion as before, set up a for loop with index i to iterate through the batch of sprites). In here paths will be moved and drawn in order to draw the walls to the canvas. Finally, after the for loop, the stroke method of canvasContext is called and this will finally draw all of the paths we pass to the context to the canvas.

```

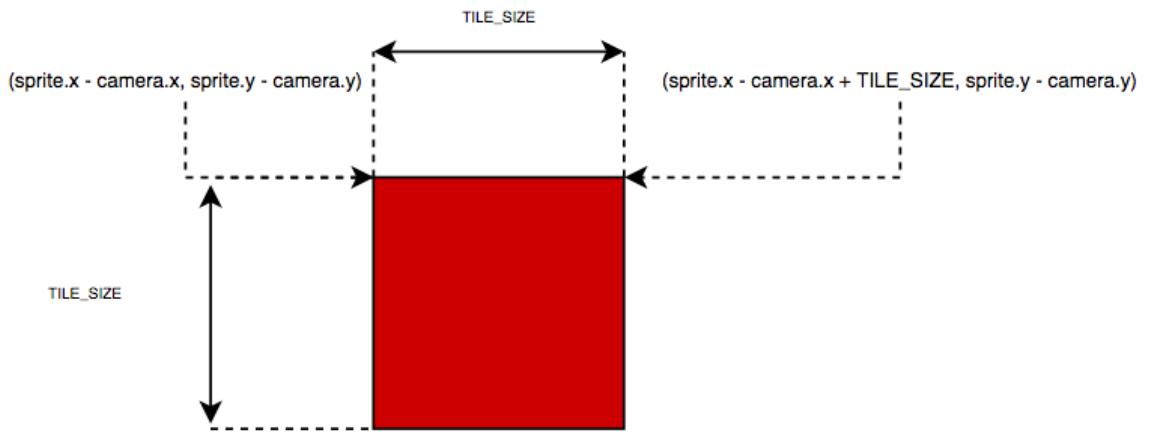
105
106      // Set the colour of the walls to the constant WALL_COLOUR
107      this.canvasContext.strokeStyle = WALL_COLOUR;
108      // Set the width of the walls to constant TILE_SIZE divided by 16
109      this.canvasContext.lineWidth = TILE_SIZE / 16;
110
111      // Begin drawing the path
112      this.canvasContext.beginPath();
113
114      // Draw the walls
115      for (var i in this.batch) {
116      }
117
118      // Draw the walls to canvasContext with .stroke() function
119      this.canvasContext.stroke();

```

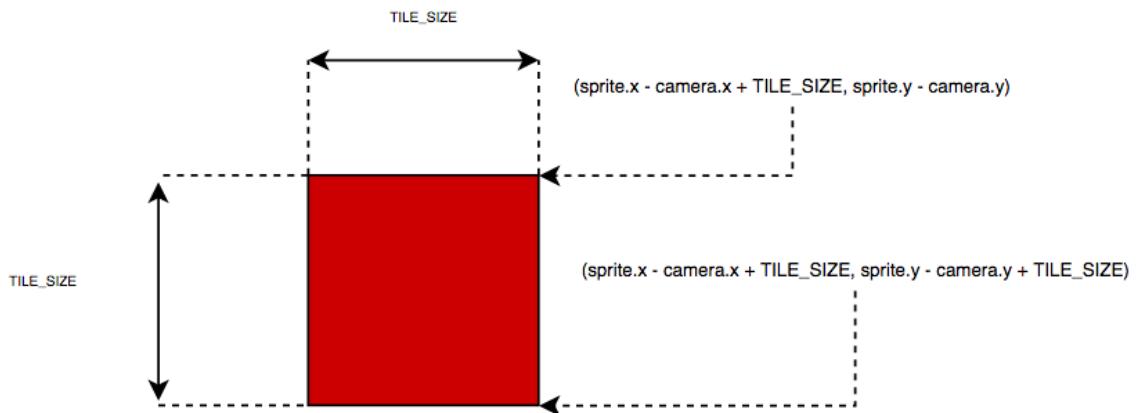
Once this had been written, it was time to flesh out the body of the for loop. Before any code was written, it was necessary to create diagrams to help describe how the coordinate geometry would work to draw the lines and paths successfully.

To draw a path onto a canvas, you need to move to a starting point and draw a line to an end point. Therefore, to draw a wall onto a sprite, you need the corner of the sprite where the wall starts and the corner of the sprite where the wall finishes. These can all be calculated if you know the pixel coordinates of the sprite (and the camera) using the `TILE_SIZE` constant. The following diagrams show an example of a sprite and the pair of coordinates from which a line is constructed. These illustrate the procedure for each of the walls.

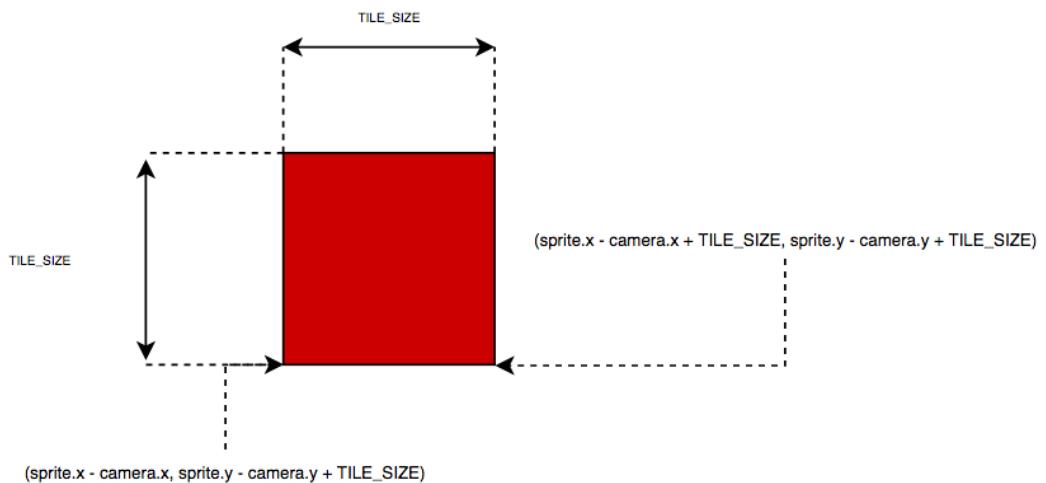
North / top wall can be drawn by drawing a line from the top left corner to the top right corner, at the coordinates shown on the diagram.



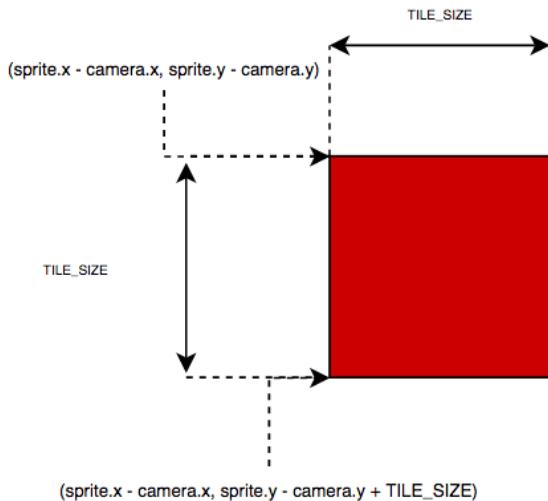
East / right wall can be drawn by drawing a line from the top right corner to the bottom right corner, as shown on the diagram below.



South / bottom wall can be drawn by drawing a line from the bottom right corner to the bottom left corner, as shown on the diagram.



The West / left wall can be drawn by drawing a line from the bottom left corner to the top left corner, as shown in the diagram below.



Now the code could be written that addresses these geometry considerations. An if statement was nested within the for loop that repeated the check that the sprite at index i of batch being drawn was within the bounds of the canvas. This was another necessary move to ensure strong performance.

Another four independent if statements were added sequentially, nested within the if statement that checks the bounds of the canvas. Each one checks that one of the Boolean values that represents a wall within the array walls of the sprite at index i of batch is true – it checks which walls are present so only they are drawn. Because the values are Boolean they can be evaluated within an if statement without the need to make any form of logical comparison (they are a logical comparison in themselves).

```
// Draw the walls
for (var i in this.batch) {
    // Check that the i'th sprite is located
    // within the bounds of the camera
    // This means computer resources aren't wasted rendering
    // a sprite that isn't going to be seen

    // Subtracting the camera's coordinate's from those of
    // the sprite effectively moves the sprites with respect
    // to a stationary camera. This has the effect of the camera
    // moving and the sprites remaining stationary
    if ((this.batch[i].x - this.camera.x < this.camera.width) &&
        (this.batch[i].y - this.camera.y < this.camera.height)) {

        // Check if there is a top / north wall and draw if there is
        if (this.batch[i].walls[0]) {
        }

        // Check if there is a right / east wall and draw if there is
        if (this.batch[i].walls[1]) {
        }

        // Check if there is a bottom / south wall and draw if there is
        if (this.batch[i].walls[2]) {
        }

        // Check if there is a left / west wall and draw if there is
        if (this.batch[i].walls[3]) {
        }
    }
}
```

Finally, the code is added that will move the starting point of the paths and draw lines. This done through two methods of canvasContext – movePath() and drawPath(). Both take two arguments, the x and y coordinates of a pixel. For each of the walls, the canvasContext moves to the starting coordinates of where the line should be drawn then moves to the end coordinates. Refer to the previous diagrams to see the exact coordinates that will be used. This code is shown over two images overleaf.

```
127
128     // Check if there is a top / north wall and draw if there is
129     if (this.batch[i].walls[0]) {
130         // Move the starting point to the top left-hand corner of ith sprite
131         this.canvasContext.moveTo(
132             this.batch[i].x - this.camera.x,
133             this.batch[i].y - this.camera.y
134         );
135
136         // Draw a line to the top right-hand corner of ith sprite
137         this.canvasContext.lineTo(
138             this.batch[i].x - this.camera.x + TILE_SIZE,
139             this.batch[i].y - this.camera.y
140         );
141     }
142
143     // Check if there is a right / east wall and draw if there is
144     if (this.batch[i].walls[1]) {
145         // Move to the top-right hand corner of ith sprite
146         this.canvasContext.moveTo(
147             this.batch[i].x - this.camera.x + TILE_SIZE,
148             this.batch[i].y - this.camera.y
149         );
150
151         // Draw a line to the bottom-right hand corner of ith sprite
152         this.canvasContext.lineTo(
153             this.batch[i].x - this.camera.x + TILE_SIZE,
154             this.batch[i].y - this.camera.y + TILE_SIZE
155         );
156     }
157
158     // Check if there is a bottom / south wall and draw if there is
159     if (this.batch[i].walls[2]) {
160         // Move to the bottom-right hand corner of ith sprite
161         this.canvasContext.moveTo(
162             this.batch[i].x - this.camera.x + TILE_SIZE,
163             this.batch[i].y - this.camera.y + TILE_SIZE
164         );
165
166         // Draw a line to the bottom-left hand corner of ith sprite
167         this.canvasContext.lineTo(
168             this.batch[i].x - this.camera.x,
169             this.batch[i].y - this.camera.y + TILE_SIZE
170         );
171     }
172
173     // Check if there is a left / west wall and draw if there is
174     if (this.batch[i].walls[3]) {
175         // Move to the bottom-left hand corner of ith sprite
176         this.canvasContext.moveTo(
177             this.batch[i].x - this.camera.x,
178             this.batch[i].y - this.camera.y + TILE_SIZE
179         );
180
181         // Draw a line to the top-left hand corner of ith sprite
182         this.canvasContext.lineTo(
183             this.batch[i].x - this.camera.x,
184             this.batch[i].y - this.camera.y
185         );
186     }
187 }
188 }
```

### Second alpha test of the draw function

With the wall drawing code added the draw function is completed and it is time to alpha test these new additions. The aim of this alpha testing is to ensure that the walls are rendered correctly. The test code that was written earlier was slightly modified and is reproduced below.

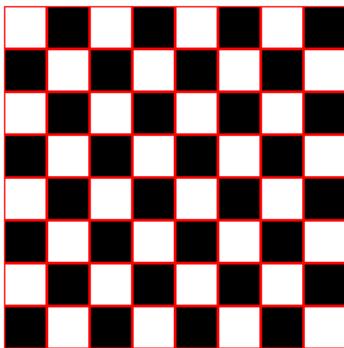
There are multiple modifications to this test code – note the inclusion of constant WALL\_COLOUR on line 198. This is included as the constants have not yet been defined (see engine.js later in this section) and they are required for the SpriteBatch code to function so are temporarily inserted here. Note also on line 219 that within the ternary decision operator that the colours have changed from red and blue to black and white – this is to provide a greater contrast with red walls.

```

196 // Initialise a temporary constant for tile size
197 const TILE_SIZE = 32;
198 const WALL_COLOUR = "red";
199
200 // Get the canvas drawing context
201 var canvas = document.getElementById('canvas');
202
203 canvas.width = 512;
204 canvas.height = 512;
205
206 var canvasContext = canvas.getContext('2d');
207
208 // Instantiate a new SpriteBatch
209 var spriteBatch = new SpriteBatch(canvasContext, new Camera());
210
211 var alternate = false;
212
213 // Fill it with a grid of 8 by 8 sprites
214 for (var i = 0; i < 8; i++) {
215   for (var j = 0; j < 8; j++) {
216     // Push a new sprite to the batch
217     spriteBatch.batch.push(
218       new Sprite(
219         alternate ? "black" : "white",
220         i*TILE_SIZE,
221         j*TILE_SIZE,
222         [true,true,true,true]
223       )
224     );
225     // Invert alternate
226     alternate = !alternate;
227   }
228   // Invert alternate
229   alternate = !alternate;
230 }
231
232 // Draw the sprites
233 spriteBatch.draw();
234

```

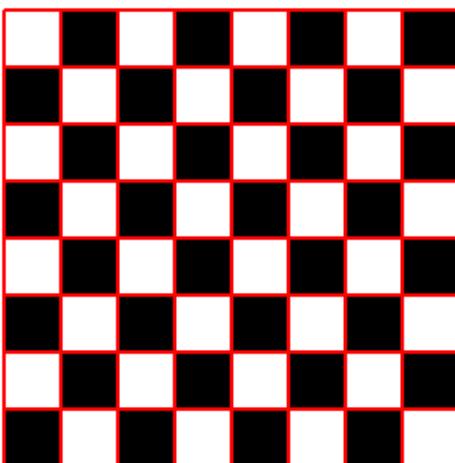
Below is the output onto the canvas in the HTML page of this test code. Note the small pixel borders of the walls. This appears to be as desired by the game, until on closer inspection there appears to be a problem.



Note that the border appears thinner on the left and topsides of the canvas. This is not have the walls are supposed to be rendered so needed to be inspected. It would appear as if the canvas was clipping off a portion of the wall. To remedy this, the test code was altered to add a translation of 10 positive pixels to the x and y coordinates of all the sprites when they are drawn. This is shown highlighted in the code snippet below.

```
// Fill it with a grid of 8 by 8 sprites
for (var i = 0; i < 8; i++) {
    for (var j = 0; j < 8; j++) {
        // Push a new sprite to the batch
        spriteBatch.batch.push(
            new Sprite(
                alternate ? "black" : "white",
                i*TILE_SIZE +10, // Shift 10 pixels right
                j*TILE_SIZE +10, // Shift 10 pixels down
                true,true,true,true
            )
        );
        // Invert alternate
        alternate = !alternate;
    }
    // Invert alternate
    alternate = !alternate;
}
```

Below is another image of the canvas's output – the translation of the canvas shows that the position of the grid so close to the edge caused it to be clipped from view. Note this will not be a problem later in the program because its size and the coordinates of the camera will dynamically decide the position of the maze.



### Timer class function

This section pertains to the development of the foundations of Output Requirements 28, 33 and 34, as all of these requirements require timing functionality that is developed here.

Timer objects need to be created so that time can be tracked and used within the context of the game. Many of the game's objects require methods that are to be called periodically or need state to change over time. Therefore, a method that maintains a timer is necessary for the game to function correctly.

To begin developing the timer, the timer.js file was opened and a multiline comment header written at the top followed by an empty function definition for Timer with no parameters.

```

1  /*
2   * timer.js
3   * Timer object
4   _____
5   Written on the 02/01/2014 by Chris Jones
6
7   This is a class function that will be used to instantiate Timer
8   objects. These represent the Timer data structures in the
9   nature of the solution section of Design.
10
11  Timers are used in a variety of methods and objects in the game to
12  keep track of timing. This is crucial to a variety of areas including
13  camera panning, player animation and the fading of the player's
14  colour.
15
16  Timer has 2 local variables:
17      - lastTime: Stores the current number of ms at the last call
18          of the run method.
19
20      - elapsedTime: Stores the amount of time that has elapsed since
21          the last call of the run method.
22
23  Timer has 1 method:
24      - run: This method will only return true when the Number parameter
25          wait_time has elapsed (wait_time is in milliseconds). To
26          call something periodically, call it with a constant wait_time
27          enclosed in an if loop that has a boolean conditional of the
28          return value of this run method.
29 */
30 function Timer() {
31     /* ... Insert functionality here ... */
32 }
```

Next, some local variables need to be declared in order to store the time changes in between run cycles. These are lastTime and elapsedTime.

```

30 function Timer() {
31
32     // Declare lastTime with the value of the time when timer is initialised
33     // Set lastTime equal to the current milliseconds on timer
34     var lastTime = new Date().getTime();
35
36     // elapsedTime stores the time taken between calls of the run method
37     // Set to zero on declaration
38     var elapsedTime = 0;
39 }
40
```

The run function was then written. This took a single parameter wait\_time, a number that represents the number of milliseconds that must have elapsed since the last call in order for it to return true.

The function first declares a local variable 'now' and stores the current time. The elapsed time is then incremented by the difference between now and lastTime. lastTime is then set to equal now.

If the elapsedTime variable's value equals or exceeds the parameter wait\_time, elpasedTime is set to zero and a value of true is returned. This in effect means elapsedTime's value is not changed until the next run when another increment takes place and another check whether it equals wait\_time.

The function is shown written into the complete Timer class function in the image below.

```
30  function Timer() {
31
32      // Declare lastTime with the value of the time when timer is initialised
33      // Set lastTime equal to the current milliseconds on timer
34      var lastTime = new Date().getTime();
35
36      // elapsedTime stores the time taken between calls of the run method
37      // Set to zero on declaration
38      var elapsedTime = 0;
39
40      // Method returns true every period of 'wait_time'
41      // For this to work this method must be polled
42      // By repeatedly calling the run method and over| given intervals,
43      // time changes can be tracked
44      this.run = function(wait_time) {
45          // Set now to current ms
46          var now = new Date().getTime();
47
48          // Add the difference between now and last check of ms to elapsedTime
49          elapsedTime += (now - lastTime);
50          // Let lastTime equal now
51          lastTime = now;
52
53          // If the elapsedTime since last poll of function
54          // is greater than the wait
55          if (elapsedTime >= wait_time) {
56              // Set elapsedTime to zero
57              elapsedTime = 0;
58              // Return a true value
59              return true;
60          }
61      }
62  }
```

### Alpha testing Timer

In order to test the timer, a small amount of test code had to be written. This was written at the end of the Timer.js file temporarily and was removed at the end of the alpha test for this file.

A new Timer object is instantiated with identifier t and a counter with a Number value of 1. The setInterval function is called, being passed an anonymous function as the first argument and 1 as the second argument. setInterval is called with a period equal to the second argument in milliseconds. This means the function passed in the first argument is called every 0.001 seconds.

Within the anonymous function's body, an if statement contains a conditional with a call to t's run method with an argument of 1000. This means this method call returns true every 1000ms (1 second). When this method returns true, the if statement is evaluated as true so its body is executed. This contains a write to the console of the value of the counter and a string concatenation. Finally the counter is incremented within the body of the if loop.

This code has the effect of outputting the number of elapsed seconds to the console one at a time by using a timer. Below is the completed code written at the bottom of timer.js. Note the commenting is minimal as this is only test code.

```

64  var t = new Timer();
65  var counter = 1;
66
67 // Run this code repeatedly ever 0.001 seconds
68▼ setInterval(function() {
69
70    // If 1 second (1000ms) has elapsed
71▼  if (t.run(1000)) {
72
73    // Output the number of elapsed seconds using a counter
74    console.log(String(counter) + " second(s)");
75
76    // Increment the counter
77    counter += 1;
78
79  }
80
81 }, 1);

```

Navigating to the game's webpage and opening the console yielded the following result shown in the screenshot below.

The screenshot shows a browser's developer tools console tab selected. The log area displays the following output:

```

1 second(s)
2 second(s)
3 second(s)
4 second(s)
5 second(s)
6 second(s)
7 second(s)
8 second(s)
9 second(s)

```

Although it cannot be seen from this view the time stamps of each console write it is apparent there are individual writes and observing with a stopwatch allowed me to note they were output every second. This means the timer works as intended and is accurate enough for use in the game.

### Player class function

*This section pertains to the development of the foundations of Output Requirements 28 and 29.*

It was time to proceed with the code for class that would model an object to represent the player. This began with opening player.js and consulting the data structures section of nature of the solution in order to glean information on how to develop this structure.

Firstly, as with all the previous JavaScript files created in the development of the software, a multiline comment header was added in order to explain the file along with an empty function definition with all necessary parameters. This is the lengthiest one yet and shown in the two images below.

```
6      This file represents the Player data structure represented in the Data
7      Structures section of the design specification.
8
9
10     The function Player represents a function from which the Player
11     object can be instantiated. This will represent the Player data
12     structure.
13
14     A player object encapsulates all of the functionality necessary with
15     introducing a player into the game. It exposes getter and setters to
16     make accessing the position of the player more expressive for later
17     programming and also ensures the player is correctly coloured and
18     animated.
19
20     A player object is instantiated by the Maze object and controlled
21     within that object's functionality. This allows it to be added to
22     a SpriteBatch, thus be drawn by that and updated by the Maze.
23
24     Player has 4 attributes:
25         - disabled: This is a boolean value that is used to control
26             whether the player can move or not. This is so
27             that the player's movement can be disabled
28             when the camera is panning from the exit tile or
29             currently on a menu.
30
31         - sprite: This is the object that is responsible for the visuals
32             and drawing of the player to the canvas. This attribute
33             will be referenced to within a SpriteBatch.
34
35         - newPositionX: This is a Number that represents an x coordinate.
36             This is used to track the exact pixel coordinates of
37             a player in the maze instead of relative to other
38             grid squares. Doing this allows smooth animations to
39             be implemented.
40
41         - newPositionY: This is a Number that represents a y coordinate.
42             This is used to track the exact pixel coordinates of
43             a player in the maze instead of relative to other
44             grid squares. Doing this allows smooth animations to
45             be implemented.
46
47     Player has 2 pairs of getter / setter methods:
48         - x: This is a getter / setter pair that reference the x coordinate
49             of the sprite. This is effectively assigning the reference
50             player.x to player.sprite.x. This makes it easier to get and
```

```

51         set values of the player's position without accessing the
52         sprite.
53     - y: This is a getter / setter pair that reference the y coordinate
54         of the sprite. This is effectively assigning the reference
55         player.y to player.sprite.y. This makes it easier to get and
56         set values of the player's position without accessing the
57         sprite.
58
59 Player has 3 methods:
60     - animatePosition: This method updates the position of the player
61         on the screen. This is so that it moves smoothly
62         from tile to tile in maze.
63
64     - fadeColour: This returns a HSL CSS string that varies with respect
65         to time. This can be used to be assigned to the colour
66         values of objects in the maze – in this case the player's
67         sprite.
68
69     - trailColour: Returns a HSL CSS string that is the colour that tiles
70         on which the player is leaving a trail should be
71         coloured.
72
73 Player has 8 local variables:
74     - hueOne: This is the initial hue of the fading colour.
75
76     - hueTwo: This is the hue to which the fading colour fades.
77
78     - currentHue: The current hue of the player at any given time.
79
80     - dHue: A change in hue. This is a unit of change in hue with one
81         step in time.
82
83     - currentSaturation: The current saturation of the fading colour.
84
85     - currentLightness: The current lightness of the fading colour.
86
87     - fadeDirection: The fading colour will fade from hueOne to hueTwo and
88         then back again in a periodic cycle. This is a boolean
89         that stores in which direction the fading is occurring.
90
91     - fadingTimer: This is a Timer object that keeps track of the timing
92         with respect to which the colour fades.
93 */
94 function Player(sprite) {
95     /* Insert functionality here */
96 }

```

Firstly, the two pairs of getter / setter methods are added. These are written expressly for the purpose of making programming using Player objects easier in the future. They allow calls to get the value and assign to the value of the x and y attributes of sprite as if they belong directly to player (one could call player.x on a Player object and it would represent player.sprite.x). It makes it appear as if the player has its own position information when in fact this functionality is retained by its sprite attribute.

These are defined using special functions `__defineGetter__` and `__defineSetter__` and passing each a function of choice. `__defineGetter__` must return a value that will be represented by the property of the string passed as the first argument (in this case “x” and “y”). Functionality is similar for `__defineSetter__`, where the function that is passed as a parameter to the anonymous function and is used to set its new value. This functionality is shown in the image below and should become more self explanatory when the code is seen.

For example, calling `player.x` where `player` is a `Player` object will execute the function passed anonymously to the `__defineGetter__` call, and in this case would return the value of `player.sprite.x`. Conversely, if `player.x = 12` was called on a `Player` object, the function passed anonymously the `__defineSetter__` call would be executed with the argument `x` equal to 12. This would then assign `player.sprite.x` the value/reference of parameter `x`, which in this case is 12. The same is true for the y getter / setter pair.

```

94▼ function Player(sprite) {
95    // Define getter and setter for x position property
96    // These make it easier to reference the position of player
97▼   this.__defineGetter__("x", function() {
98        // Return the value of the sprite's x property
99        return this.sprite.x;
100);
101▼   this.__defineSetter__("x", function(x) {
102        // Set the value of the sprite's x property to the parameter x
103        this.sprite.x = x;
104);
105
106    // Define getter and setter for y position property
107    // These make it easier to reference the position of player
108▼   this.__defineGetter__("y", function() {
109        // Return the value of the sprite's y property
110        return this.sprite.y;
111);
112▼   this.__defineSetter__("y", function(y) {
113        // Set the value of the sprite's y property to the parameter y
114        this.sprite.y = y;
115);
116}
117

```

Next the value of the attributed disabled is set to false by default. This is shown at the end of the player class function in the image below.

```

117    // Set the disabled property of the player to false by default
118    // When disabled is true movement of player is restricted
119    // This occurs in the maze.js position updating methods
120    this.disabled = false;
121}
122

```

Next we perform some validation checks on the sprite parameter before assigning it to the attribute sprite. This involves an if statement with a simple validation check that ensures the parameter sprite is an instance of a Sprite object. If this is true it is assigned to the sprite attribute. If not, an error is thrown. There is no default value in this case because the player should not default to anything. This code was written at the end of the class function and shown in the image below.

```

122    // Check if sprite parameter is of type Sprite
123▼  if (sprite instanceof Sprite) {
124      // Assign value to parameter sprite to property sprite
125      this.sprite = sprite;
126    // Otherwise throw an error
127    } else {
128      throw "Argument error. sprite must be a Sprite object."
129    }
130}
131

```

Next we assign to the attributes newPositionX and newPositionY. These are set sequal to the x and y getter methods respectively (which return this.sprite.x and this.sprite.y respectively). It can already be seen how neatly position values for the player can be accessed with the getter / setter pairs.

```

131    // We know by validation in the sprite class its properties must be correct
132    // Therefore we can assign new values here
133    this.newPositionX = this.x;
134    this.newPositionY = this.y;
135}
136

```

Next the function for animating the player's position is written. This code attempts to determine whether player's new position value is greater or less than its actual position value. If it is one of these conditions, it moves it a small increment towards this correct position value. This way, by assigning the newPositionX or newPositionY to a value in another area of code and animatePosition is called repeatedly the player will move smoothly from its current position to the new position.

This will become more evident when implemented within the update cycle of a Maze object. In order for its functionality to be meaningfully observed there needs to be a repeated call to this function.

This method is shown in the code below.

```
135 // This method updates the position of the player on the screen
136 // This is so that it moves smoothly from tile to tile in maze
137 this.animatePosition = function() {
138     // Update the x position of the sprite in small increments
139     // This ensures the player moves smoothly
140     if (this.newPositionX > this.x) {
141         // Increase x coordinate by quarter of the size of game tiles
142         this.x += TILE_SIZE/4;
143     } else if (this.newPositionX < this.x) {
144         // Decrease x coordinate by quarter of the size of tiles
145         this.x -= TILE_SIZE/4;
146     }
147
148     // Update the y position of the sprite in small increments
149     // This ensures the player moves smoothly
150     if (this.newPositionY > this.y) {
151         // Increase y coordinate by quarter of the size of game tiles
152         this.y += TILE_SIZE/4;
153     } else if (this.newPositionY < this.y) {
154         // Decrease y coordinate by quarter of the size of tiles
155         this.y -= TILE_SIZE/4;
156     }
157 }
158 }
159 }
160 }
```

The code that follows in the Player class function manages the fading of the player's colour. This is not assigned to the player's sprite natively – instead a Maze object will do this later. While this may seem counterintuitive now, it would become more apparent should the game be updated or extended in the future by another player who wants the colour mechanics of the player to differ. It in essence offers more room for expansion of the program.

This begins with the declaration of eight local variables, whose definition is described in the comment header and in the comments below – and of course in the data structures section of nature of the solution. Below is an image of these local variable declarations.

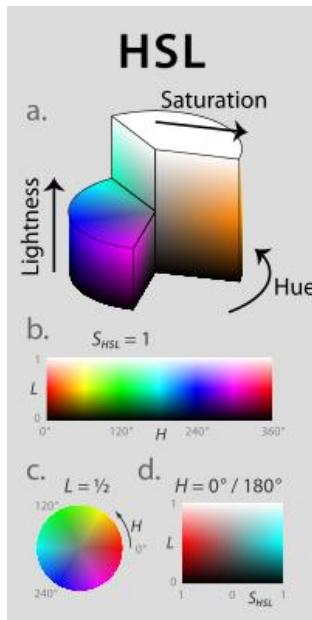
```

159     // This is the initial hue of the fading colour.
160     var hueOne = 145;
161     // This is the hue to which the fading colour fades.
162     var hueTwo = 300,
163     // The current hue of the player at any given time.
164     var currentHue = hueOne,
165     // A change in hue. This is a unit of change in hue with one step in time.
166     var dHue = (hueTwo - hueOne)/10000;
167     // The current saturation of the fading colour.
168     var currentSaturation = 80;
169     // The current lightness of the fading colour.
170     var currentLightness = 40;
171     // Stores the direction of fade of colour
172     var fadeDirection=true;
173     // Keeps track of the timing with respect to which the colour fades.
174     var fadingTimer = new Timer();
175 }
176
177

```

The fadeColour function was then written. This provides a basic mathematical ability to alter the hue component of a HSL CSS value of the sprite's colour so that it changes with respect to time. But before it could be written the HSL model needed to be understood

HSL is a system for defining the RGB colour model used on computer systems and is defined by three components – Hue, saturation and lightness. It is a cylindrical model that maps coordinates on a cylinder to values of colour. Hue can take any value between 0 and 360, representing the degrees through which the cylinder is moved. Saturation and lightness represent the height and radius of the cylinder and take values of percentage. The image below from [http://en.wikipedia.org/wiki/HSL\\_and\\_HSV](http://en.wikipedia.org/wiki/HSL_and_HSV) makes it easier to see diagrammatically how it works.

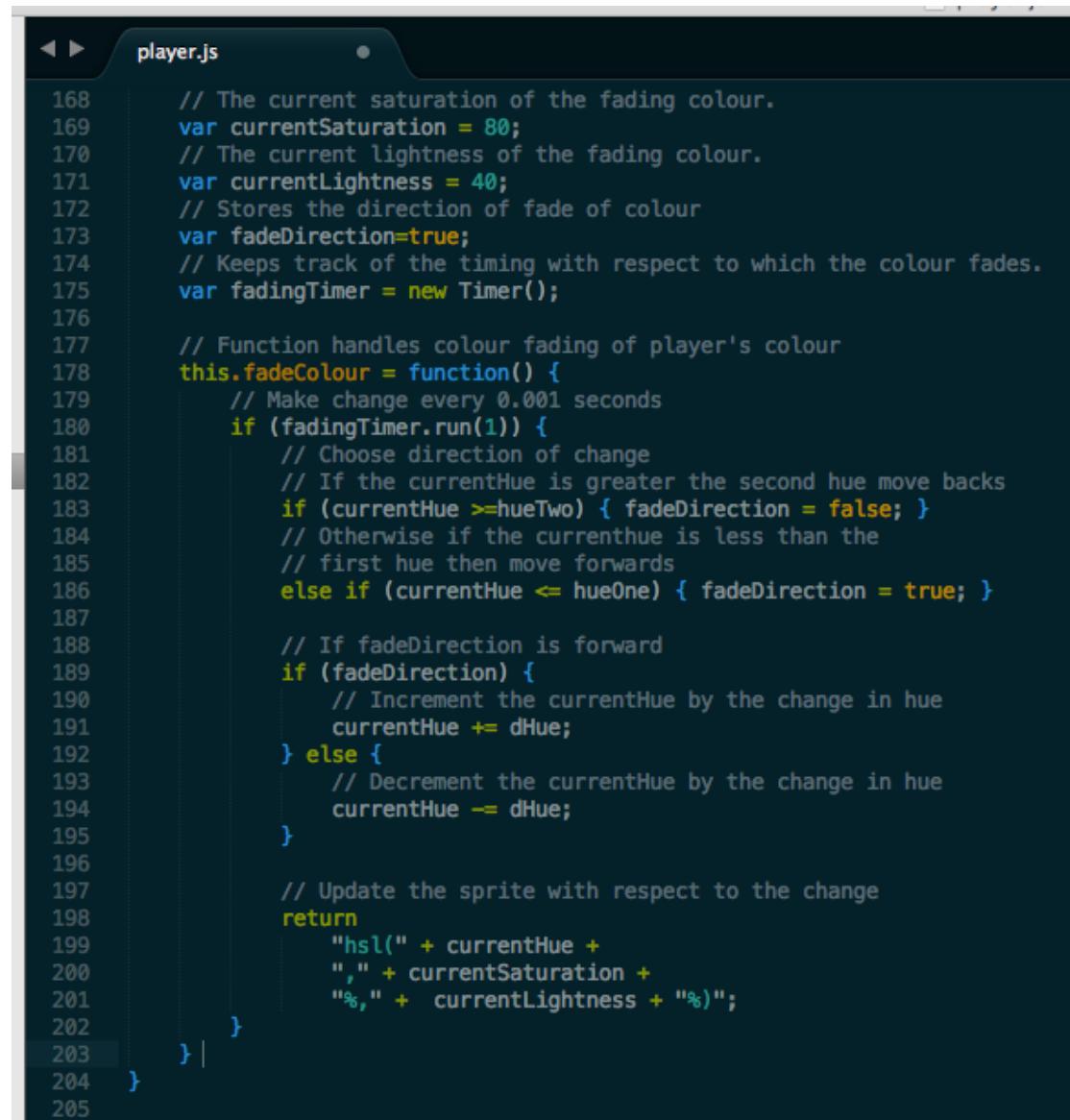


HSL is being used for the fadeColour function because the hue, which is the component that will change during colour fade, is encapsulated in one component of HSL (rather than all three components of RGB, for example) which means lightness and saturation can remain unchanged.

There are two hues between which the colour fades in a periodic cycle – hueOne and hueTwo. The saturation and lightness remain the same but are not declared as constants in case they need to be changed during development. The currentHue is the hue that takes a value between hueOne and hueTwo inclusive. dHue is a single step of change in hue and currentHue moves between hueOne and hueTwo in steps of dHue. As can be seen in the image above of the declaration of the local

variables, dHue is declared as a ten thousandth of the difference between hueTwo and hueOne. fadingTimer will be used to ensure the fading logic only occurs once every 0.001 seconds. fadingDirection is used to ensure that the correct change in hue occurs depending on whether the colour is fading from hueOne to hueTwo or from hueTwo to hueOne.

Below is an image of the code that was written to facilitate this. At the end a string is concatenated with the current values of hue, saturation and lightness into one that is compliant with CSS standards and will be understood by the canvas.

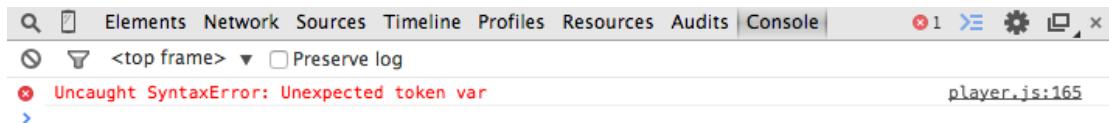


```
player.js
168     // The current saturation of the fading colour.
169     var currentSaturation = 80;
170     // The current lightness of the fading colour.
171     var currentLightness = 40;
172     // Stores the direction of fade of colour
173     var fadeDirection=true;
174     // Keeps track of the timing with respect to which the colour fades.
175     var fadingTimer = new Timer();
176
177     // Function handles colour fading of player's colour
178     this.fadeColour = function() {
179         // Make change every 0.001 seconds
180         if (fadingTimer.run(1)) {
181             // Choose direction of change
182             // If the currentHue is greater the second hue move backs
183             if (currentHue >=hueTwo) { fadeDirection = false; }
184             // Otherwise if the currentHue is less than the
185             // first hue then move forwards
186             else if (currentHue <= hueOne) { fadeDirection = true; }
187
188             // If fadeDirection is forward
189             if (fadeDirection) {
190                 // Increment the currentHue by the change in hue
191                 currentHue += dHue;
192             } else {
193                 // Decrement the currentHue by the change in hue
194                 currentHue -= dHue;
195             }
196
197             // Update the sprite with respect to the change
198             return
199                 "hsl(" + currentHue +
200                 "," + currentSaturation +
201                 "%," + currentLightness + "%)";
202         }
203     }
204 }
```

With this the code for the player class function was completed.

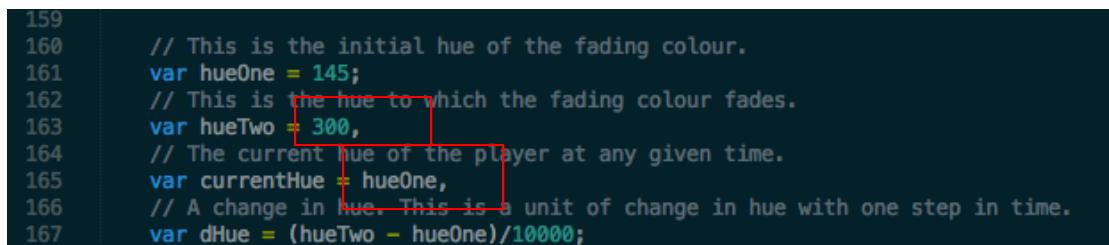
### Alpha testing the player class function

To begin alpha testing, I navigated to the index.html page of the game in Google Chrome and opened the console in order to ensure I could observe output after writing some test code. However, upon opening the console, there was an error that had already been thrown. This is shown in a screenshot below.



A screenshot of the Google Chrome DevTools Console. The tabs at the top are Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Console tab is selected. Below the tabs, there is a search bar and a dropdown menu set to <top frame>. A checkbox labeled 'Preserve log' is unchecked. A red error message 'Uncaught SyntaxError: Unexpected token var' is displayed, followed by a greater than sign '>'. To the right of the message, the file 'player.js' and line number '165' are indicated.

I noted down the line number on which the error was thrown and the type of error (in this case a syntax error, and consulted player.js, moving to line 165. This code is shown reproduced in an image below. Being a syntax error, I was looking for typos or spelling mistakes that had caused the JavaScript engine to be unable to understand the line. I realised that two of the statements on lines 163 and 165 were terminated with commas instead of the syntactically correct semicolon. Although only one error was thrown it would appear either could have been responsible and it was important to simply correct both at that point. Below are the two syntax errors shown highlighted with red boxes.



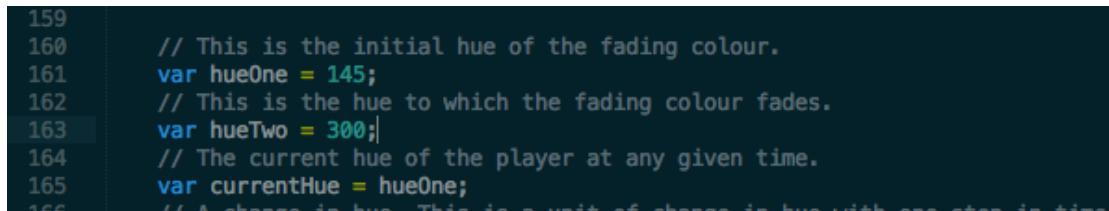
```

159
160      // This is the initial hue of the fading colour.
161      var hueOne = 145;
162      // This is the hue to which the fading colour fades.
163      var hueTwo = 300,
164      // The current hue of the player at any given time.
165      var currentHue = hueOne,
166      // A change in hue. This is a unit of change in hue with one step in time.
167      var dHue = (hueTwo - hueOne)/10000;

```

The code snippet shows lines 159 through 167. Lines 163 and 165 contain commas at the end of the statements, which are highlighted with red boxes.

The error was corrected by replacing the two commas with semicolons, shown in the image below. Finally, in the image proceeding that is a screenshot of the console after a page refresh; absent of the syntax error.

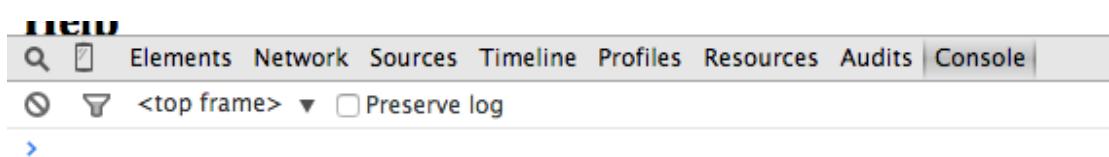


```

159
160      // This is the initial hue of the fading colour.
161      var hueOne = 145;
162      // This is the hue to which the fading colour fades.
163      var hueTwo = 300;
164      // The current hue of the player at any given time.
165      var currentHue = hueOne;
166      // A change in hue. This is a unit of change in hue with one step in time.

```

The code snippet shows lines 159 through 166. The commas from the previous version have been replaced by semicolons.



A screenshot of the Google Chrome DevTools Console after a page refresh. The tabs, search bar, and dropdown menu are identical to the first screenshot. The console is now empty, showing only the greater than sign '>'.

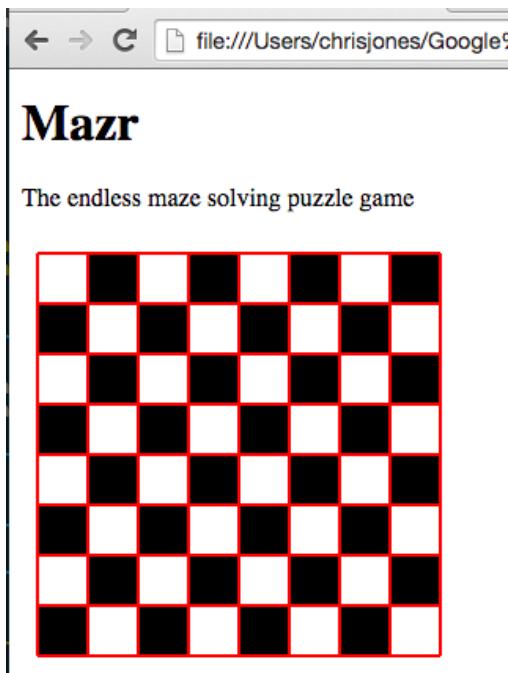
With this syntax error solved further alpha testing of the player could take place.

**Alpha testing the Player class function**

Proper alpha testing could now take place for the Player class function. In order to achieve this I copied the test code that was used in the SpriteBatch.js to the player.js file. The code appended to the end of player.js is shown in the image below.

```
209 // Initialise a temporary constant for tile size
210 const TILE_SIZE = 32;
211 const WALL_COLOUR = "red";
212
213 // Get the canvas drawing context
214 var canvas = document.getElementById('canvas');
215
216 canvas.width = 512;
217 canvas.height = 512;
218
219 var canvasContext = canvas.getContext('2d');
220
221 // Instantiate a new SpriteBatch
222 var spriteBatch = new SpriteBatch(canvasContext, new Camera());
223
224 var alternate = false;
225
226 // Fill it with a grid of 8 by 8 sprites
227 for (var i = 0; i < 8; i++) {
228    for (var j = 0; j < 8; j++) {
229        // Push a new sprite to the batch
230        spriteBatch.batch.push(
231            new Sprite(
232                alternate ? "black" : "white",
233                i*TILE_SIZE +10, // Shift 10 pixels right
234                j*TILE_SIZE +10, // Shift 10 pixels down
235                [true,true,true,true]
236            )
237        );
238        // Invert alternate
239        alternate = !alternate;
240    }
241    // Invert alternate
242    alternate = !alternate;
243 }
244
245
246 // Draw the sprites
247 spriteBatch.draw();
248
```

As a reminder of the output to the webpage that this test code produces, an image showing the canvas has been included below.



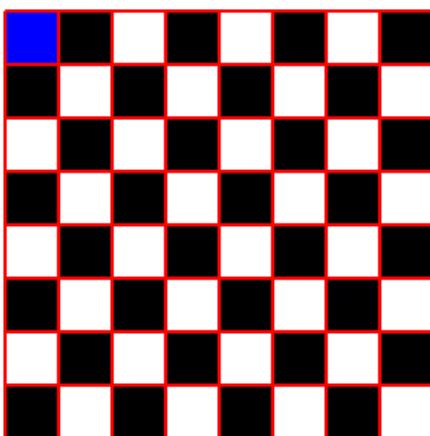
In order to alpha test the player class function, a player object will be instantiated as a Player and added to the SpriteBatch. Then, some rudimentary timing facility will be added to ensure the colour of the player fades correctly. The following code is added before calling the draw method on the spriteBatch object.

```
248 var player = new Player(new Sprite("blue", 10, 10, [false, false, false, false]));
249 spriteBatch.batch.push(player.sprite);
250
251 // Draw the sprites
252 spriteBatch.draw();
253
```

Below is an image that shows a section of the webpage. Note the blue square is the player.

## Mazr

The endless maze solving puzzle game



Now it needs to be tested as to whether the player will move appropriately. The call to draw on spriteBatch was placed in an anonymous function passed as the first argument to the setInterval

function. This is a function that is executed 30 times a second (as dictated by the second argument). A call to animatePosition of the player is also placed in this anonymous function. This has the effect of repeating this code 30 times a second.

```
var player = new Player(new Sprite("blue", 10, 10, [false,false,false,false]));
spriteBatch.batch.push(player.sprite);

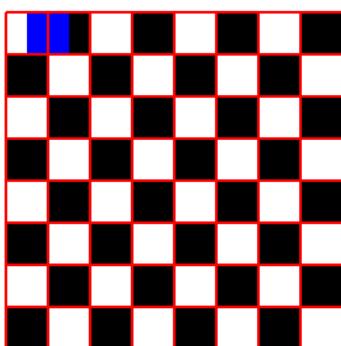
setInterval(function() {
    player.newPositionX += TILE_SIZE;
}, 1000);

setInterval(function() {
    player.animatePosition();
    spriteBatch.draw();
}, 1000/30);
```

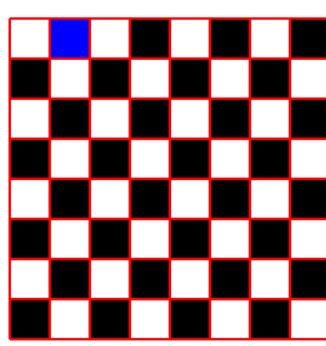
There is another setInterval, executed once every second (as the second argument is 1000 milliseconds). The newPositionX attribute of player has the value of TILE\_SIZE added to its current value. This moves the player one tile along in the grid. Therefore, the player should move across one tile every second. Below are various images that show the player moving along the maze.

**Mazr**

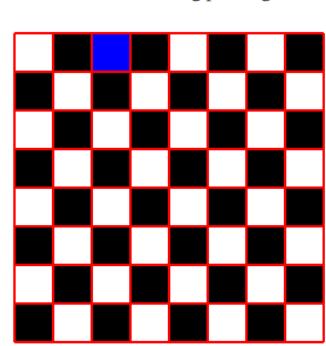
The endless maze solving puzzle game

**Mazr**

The endless maze solving puzzle game

**Mazr**

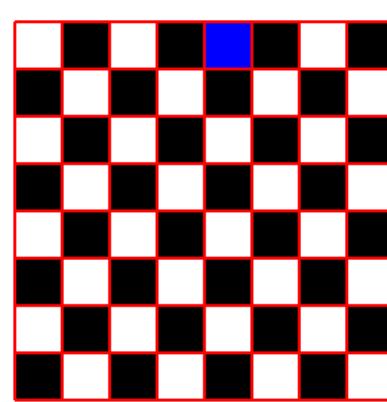
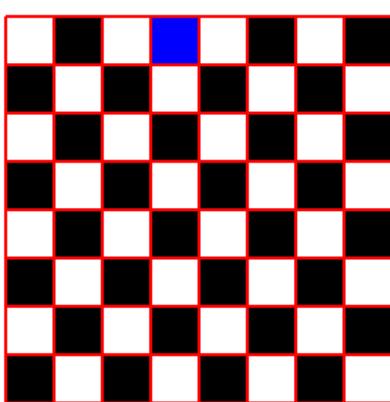
The endless maze solving puzzle game

**Mazr**

The endless maze solving puzzle game

**Mazr**

The endless maze solving puzzle game



Note that the colour fading of the player needs to also be tested but is difficult to set up in an alpha testing environment. Therefore it will be rigorously tested in the testing section later.

Next, to test the fading colour, the movement code was commented out and a new line of code added. Within the anonymous function passed to setInterval that is executed every second 30 times, the colour attribute of the sprite attribute of player is set equal to the return value of a call to

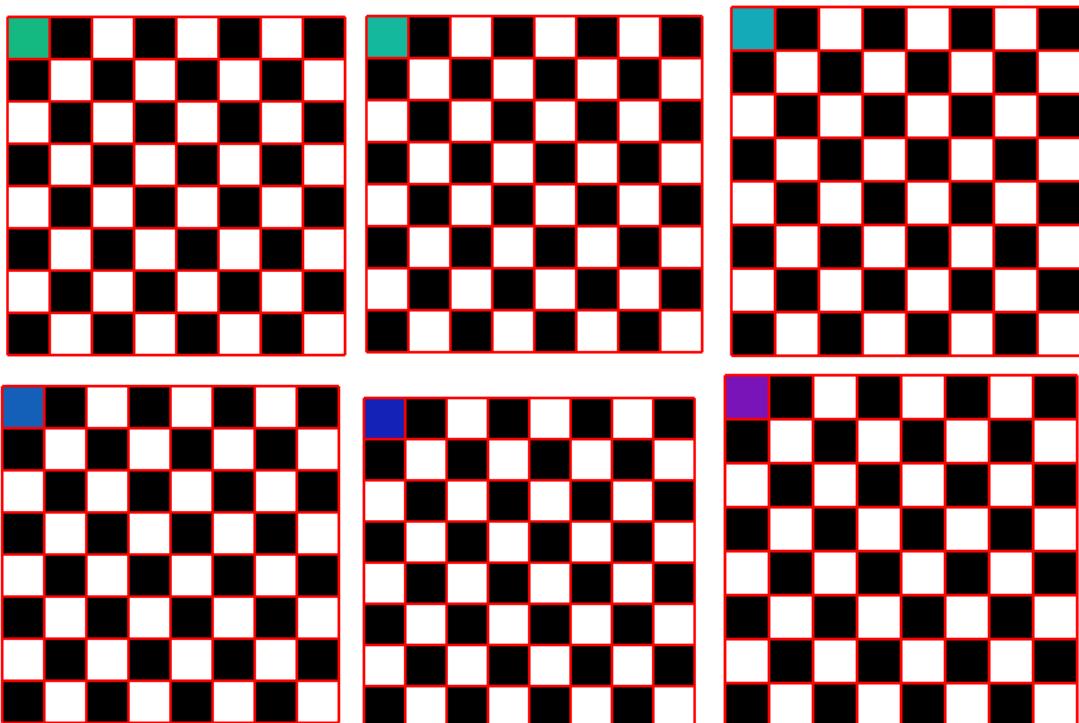
the fadeColour method of player. This has the effect of updating the colour of the player's sprite 30 times a second to the correct fading colour.

```
var player = new Player(new Sprite("blue", 10, 10, [false, false, false, false]));
spriteBatch.batch.push(player.sprite);

//setInterval(function() {
//    player.newPositionX += TILE_SIZE;
//}, 1000);

setInterval(function() {
    player.animatePosition();
    player.sprite.colour = player.fadeColour();
    spriteBatch.draw();
}, 1000/30);
```

Below are images taken within time intervals that show the colour of the player fades.



Therefore we can consider that the colour of the player fades correctly. More rigorous tests will be performed later in the Testing section. The test code was then deleted from the file now that the alpha testing for the player had been completed.

### ExitTile class function

*This section pertains to the development of the foundations of Output Requirement 30.*

An ExitTile object is required in order to store a sprite that represents the exit of the maze. This is the tile the player must move onto in order to complete the maze. It is not a complex function and as per usual development began with a multiline comment header and an empty function definition with the necessary parameters, which is shown in the image below.

```

1  /*
2   *      ExitTile.js
3   *      ExitTile object
4   *
5   *      Written on the 01/10/2014 by Chris Jones
6   *
7   *      This file represents the ExitTile data structure represented in the Data
8   *      Structures section of the design specification.
9   *
10  *      The function ExitTile represents a function from which the ExitTile
11  *      object can be instantiated. This will represent the ExitTile data
12  *      structure.
13  *
14  *      An ExitTile represents the end of a maze – the tile to which the player
15  *      must move in order to complete any given maze. Therefore, it must be
16  *      drawn to the maze as would any sprite.
17  *
18  *      An ExitTile object is instantiated by the Maze object and controlled
19  *      within the body of that object. This allows for it to be controlled and
20  *      its position used in conjunction with the Player's in order to determine
21  *      if the maze has been completed
22  *
23  *      ExitTile has 1 attribute:
24  *          - sprite: The sprite object that is drawn to the canvas that
25  *                  represents the ExitTile in the maze.
26  *
27  *      ExitTile has 2 pairs of getter / setter methods:
28  *          - x: This is a getter / setter pair that references the x coordinate
29  *              of the sprite. This is effectively assigning the reference
30  *              ExitTile.x to ExitTile.sprite.x. This makes it easier to get and
31  *              set values of the ExitTile's position without accessing the
32  *              sprite.
33  *          - y: This is a getter / setter pair that references the y coordinate
34  *              of the sprite. This is effectively assigning the reference
35  *              ExitTile.y to ExitTile.sprite.y. This makes it easier to get and
36  *              set values of the ExitTile's position without accessing the
37  *              sprite.
38  */
39  function ExitTile(sprite) {
40      /* Insert functionality here */
41  }
42

```

There is a small amount of functionality for an ExitTile – it is effectively a named wrapper for a Sprite that is semantically associated with the end of a maze. In the same style as that of a player it has with it two pairs of getter / setter methods which will make it simpler to program later for the Maze class function.

These are included with little explanation below. Look back at the player development section for more information on getters / setters – these are implemented in identical fashion to the ones for the player and the code for which was copied directly from the player.js file. Below is an image showing these getters / setters entered into the file.

```
39  function ExitTile(sprite) {
40      // Define getter and setter for x position property
41      // These make it easier to reference the position of an ExitTile
42      this._defineGetter_("x", function() {
43          // Return the value of the sprite's x property
44          return this.sprite.x;
45      });
46      this._defineSetter_("x", function(x) {
47          // Set the value of the sprite's x property to the parameter x
48          this.sprite.x = x;
49      });
50
51      // Define getter and setter for y position property
52      // These make it easier to reference the position of an ExitTile
53      this._defineGetter_("y", function() {
54          // Return the value of the sprite's y property
55          return this.sprite.y;
56      });
57      this._defineSetter_("y", function(y) {
58          // Set the value of the sprite's y property to the parameter y
59          this.sprite.y = y;
60      });
61  }
62
63
```

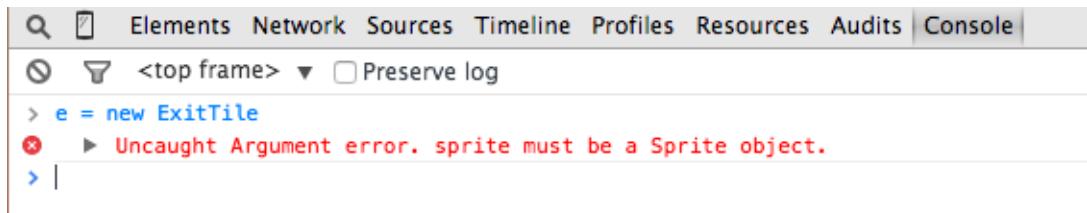
The only attribute is the sprite that belongs to the exit tile. This is implemented in the usual if-statement mode in order to validate the parameter of the method.

```
62  // Check if sprite parameter is of type Sprite
63  if (sprite instanceof Sprite) {
64      // Assign value to parameter sprite to property sprite
65      this.sprite = sprite;
66      // Otherwise throw an error
67  } else {
68      throw "Argument error. sprite must be a Sprite object."
69  }
70 }
71
```

With that simple functionality the ExitTile is completed and some simple alpha testing can begin.

### Alpha testing exit tile

The exit tile is a simple function that will require minimal testing. Firstly, I visited the webpage on which the game is embedded and entering the JavaScript console. A line is entered which instantiates a variable e as a new ExitTile with no arguments. This throws an error, as expected, which is shown below.



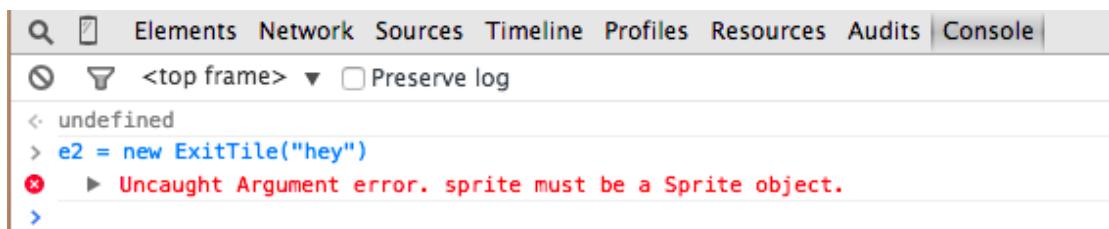
A screenshot of a browser's developer tools Console tab. The tab bar includes Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Console tab is active. The log shows the following:

```

> e = new ExitTile
✖ ▶ Uncaught Argument error. sprite must be a Sprite object.
>

```

Next, an argument is passed to the new ExitTile that is of the wrong datatype – a string. This produces another appropriate error, which is desirable.



A screenshot of a browser's developer tools Console tab. The tab bar includes Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Console tab is active. The log shows the following:

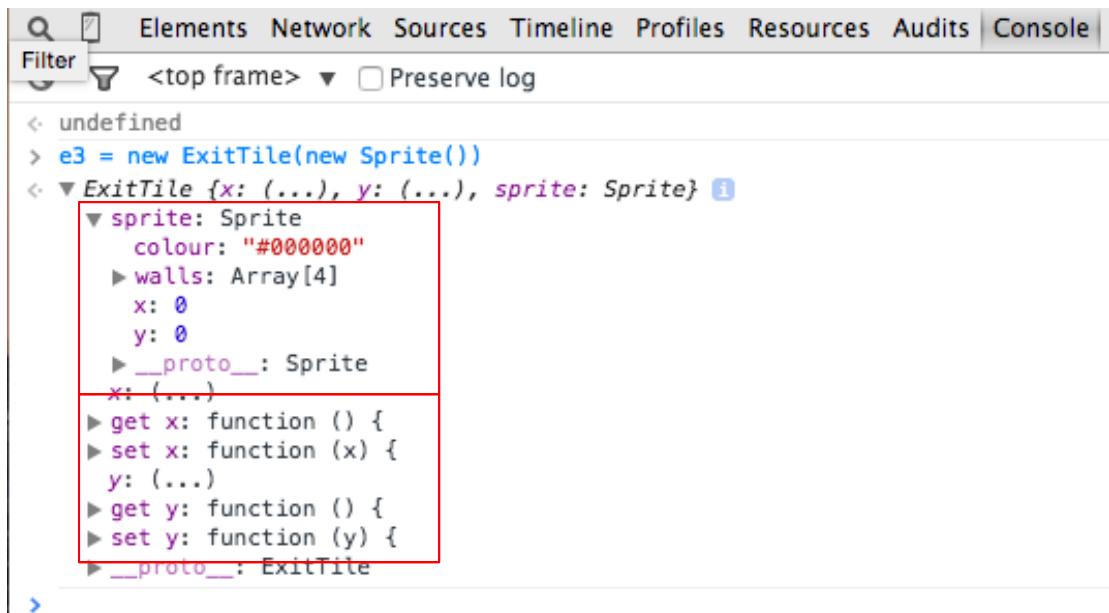
```

< undefined
> e2 = new ExitTile("hey")
✖ ▶ Uncaught Argument error. sprite must be a Sprite object.
>

```

Next, another ExitTile object e3 is instantiated and this time with an argument of a new Sprite object. Below is an image of the output this instruction produces. Note the inclusion of a new sprite object that contains default-valued attributes that would be expected from a Sprite with no arguments.

Also, the getter and setter methods are shown present. This is desirable behaviour.



A screenshot of a browser's developer tools Console tab. The tab bar includes Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Console tab is active. The log shows the following:

```

< undefined
> e3 = new ExitTile(new Sprite())
< ExitTile {x: (...), y: (...), sprite: Sprite} ⓘ
  ▼ sprite: Sprite
    colour: "#000000"
    ▶ walls: Array[4]
      x: 0
      y: 0
      ▶ __proto__: Sprite
        x: ...
      ▶ get x: function () {
      ▶ set x: function (x) {
        y: ...
      ▶ get y: function () {
      ▶ set y: function (y) {
        ▶ __proto__: ExitTile
      >

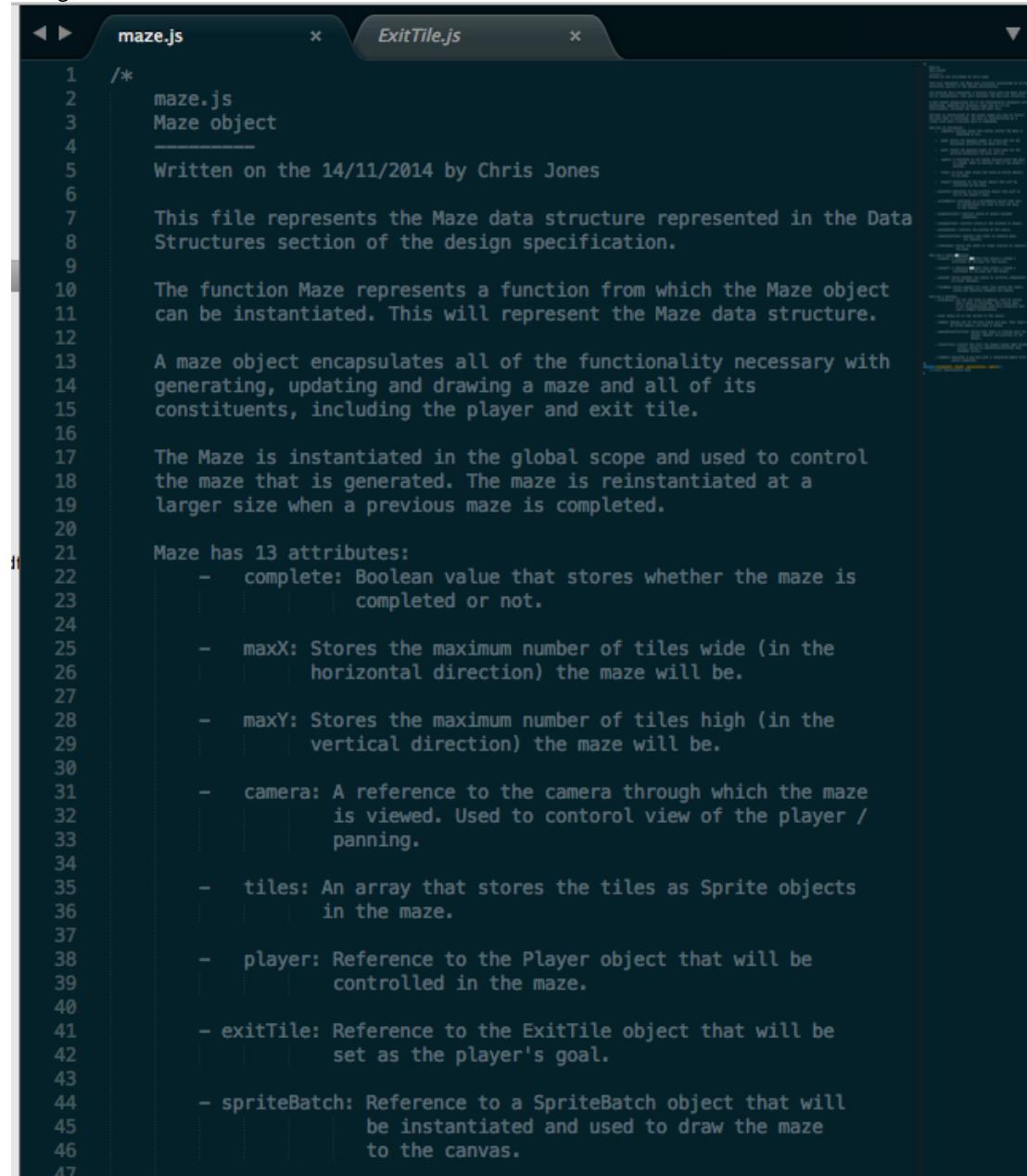
```

A red box highlights the object's properties and methods.

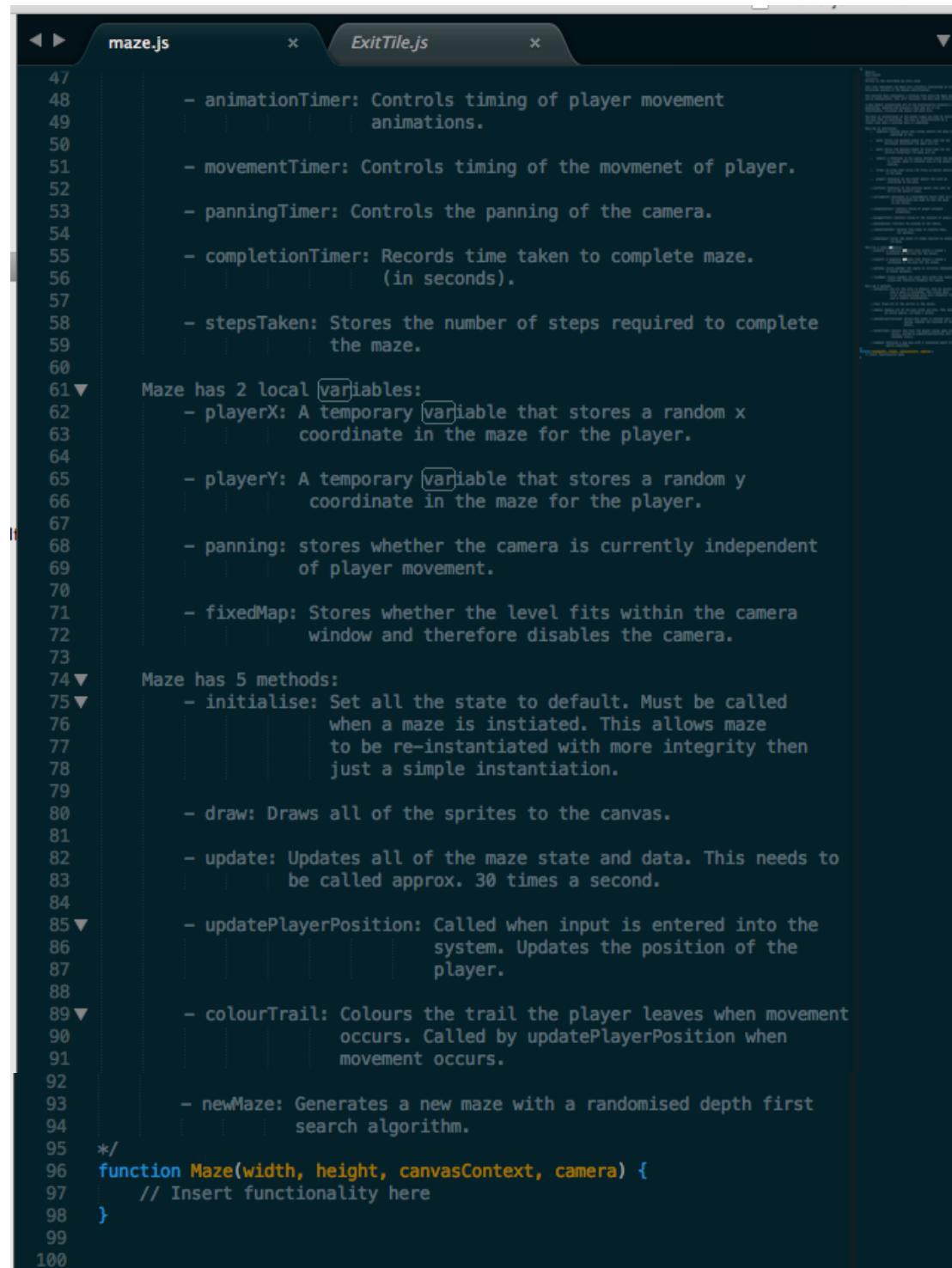
**Maze class function**

This section pertains to the development of Processing Requirements 10, 11, 12, 13, 14 and 18 and Output Requirements 30, 32, 33 and 34.

To begin the development of the Maze class function, a multiline comment header was added to describe the code that needs to be written. This file is very large so a large comment header was added to describe the functionality. This is shown in the two images below. The file also contains an empty Maze class function definition with the parameters present so that the function can begin being written.



```
/*
 1  *          maze.js
 2  *          Maze object
 3  *
 4  *          _____
 5  *          Written on the 14/11/2014 by Chris Jones
 6  *
 7  *          This file represents the Maze data structure represented in the Data
 8  *          Structures section of the design specification.
 9  *
10 *          The function Maze represents a function from which the Maze object
11 *          can be instantiated. This will represent the Maze data structure.
12 *
13 *          A maze object encapsulates all of the functionality necessary with
14 *          generating, updating and drawing a maze and all of its
15 *          constituents, including the player and exit tile.
16 *
17 *          The Maze is instantiated in the global scope and used to control
18 *          the maze that is generated. The maze is reinstantiated at a
19 *          larger size when a previous maze is completed.
20 *
21 *          Maze has 13 attributes:
22 *          - complete: Boolean value that stores whether the maze is
23 *                      completed or not.
24 *
25 *          - maxX: Stores the maximum number of tiles wide (in the
26 *                  horizontal direction) the maze will be.
27 *
28 *          - maxY: Stores the maximum number of tiles high (in the
29 *                  vertical direction) the maze will be.
30 *
31 *          - camera: A reference to the camera through which the maze
32 *                      is viewed. Used to control view of the player /
33 *                      panning.
34 *
35 *          - tiles: An array that stores the tiles as Sprite objects
36 *                      in the maze.
37 *
38 *          - player: Reference to the Player object that will be
39 *                      controlled in the maze.
40 *
41 *          - exitTile: Reference to the ExitTile object that will be
42 *                      set as the player's goal.
43 *
44 *          - spriteBatch: Reference to a SpriteBatch object that will
45 *                      be instantiated and used to draw the maze
46 *                      to the canvas.
47 *
```



```

47
48     - animationTimer: Controls timing of player movement
49             animations.
50
51     - movementTimer: Controls timing of the movmenet of player.
52
53     - panningTimer: Controls the panning of the camera.
54
55     - completionTimer: Records time taken to complete maze.
56             (in seconds).
57
58     - stepsTaken: Stores the number of steps required to complete
59             the maze.
60
61▼ Maze has 2 local variables:
62     - playerX: A temporary variable that stores a random x
63             coordinate in the maze for the player.
64
65     - playerY: A temporary variable that stores a random y
66             coordinate in the maze for the player.
67
68     - panning: stores whether the camera is currently independent
69             of player movement.
70
71     - fixedMap: Stores whether the level fits within the camera
72             window and therefore disables the camera.
73
74▼ Maze has 5 methods:
75▼     - initialise: Set all the state to default. Must be called
76             when a maze is instiated. This allows maze
77             to be re-instantiated with more integrity then
78             just a simple instantiation.
79
80     - draw: Draws all of the sprites to the canvas.
81
82     - update: Updates all of the maze state and data. This needs to
83             be called approx. 30 times a second.
84
85▼     - updatePlayerPosition: Called when input is entered into the
86             system. Updates the position of the
87             player.
88
89▼     - colourTrail: Colours the trail the player leaves when movement
90             occurs. Called by updatePlayerPosition when
91             movement occurs.
92
93     - newMaze: Generates a new maze with a randomised depth first
94             search algorithm.
95 */
96 function Maze(width, height, canvasContext, camera) {
97     // Insert functionality here
98 }
99
100

```

Next, all of the attributes that do not require validation are entered into the file. They do not require validation because they are generated by code within the class definition (and therefore by definition cannot contain invalid data passed by parameter). This is shown on the image overleaf.

```
96  function Maze(width, height, canvasContext, camera) {
97      // Boolean stores whether grid is finished
98      // Set to false by default
99      this.complete = false;
100
101     // Array of tiles that make up grid
102     this.tiles = new Array();
103
104     // Instantiate a new player attribute
105     // at the random position (playerX,playerY)
106     this.player = new Player(
107         new Sprite("hsl(145, 80%, 40%)",
108             playerX,
109             playerY,
110             [false,false,false,false]
111         )
112     );
113
114     // Set the maze's exit initially to null
115     // This is instantiated properly in the init() method
116     this.exitTile = null;
117
118     // The sprite batch that will draw the grid
119     this.spriteBatch = new SpriteBatch(canvasContext, this.camera);
120
121     // Various timer objects used to control time-dependent updates
122     this.animationTimer = new Timer(); // Controls all animations
123     this.movementTimer = new Timer(); // Controls movement of player
124     this.panningTimer = new Timer(); // Controls panning of camera
125     this.completionTimer = new Timer(); // Records time taken to complete maze
126
127     // Stores the number of seconds taken to complete the maze
128     this.completedSeconds = 0;
129
130     // Stores the number of steps taken in the maze since it has begun
131     this.stepsTaken = 0;
132 }
133 }
```

Next, the attributes that require validation are written to the file. This involves a style that is similar to previous class functions created in the development. If else statements are used to make validation checks on parameter values before the attributes are assigned. These attributes and associated validation is shown in the image below.

```

132     // Check if the width supplied is of primitive type Number
133     if (typeof width === "number") {
134         // Assign value of parameter width to attribute maxX
135         this.maxX = width;
136     }
137     // Check if no parameter was passed (undefined) or a null parameter was passed
138     else if (width === undefined || width === null) {
139         // Assign maxX attribute default value of 8
140         this.maxX = 8;
141     }
142     else {
143         // Throw an error if above conditions are not met.
144         // Note errors will be caught in the main game loop.
145         throw "Argument error. width must a number."
146     }
147
148     // Check if the height supplied is of primitive type Number
149     if (typeof height === "number") {
150         // Assign value of parameter width to attribute maxY
151         this.maxY = height;
152     }
153     // Check if no parameter was passed (undefined) or a null parameter was passed
154     else if (height === undefined || height === null) {
155         // Assign maxY attribute default value of 8
156         this.maxY = 8;
157     }
158     else {
159         // Throw an error if above conditions are not met.
160         // Note errors will be caught in the main game loop.
161         throw "Argument error. height must a number."
162     }
163
164     // Reference to camera through which maze is viewed
165     // Note there is no defaulting code path for the camera
166     if (camera === undefined || camera === null) {
167         throw "Argument error. Improper camera specified.";
168     // Otherwise correctly assign parameter to attribute
169     } else {
170         // Assign camera parameter to camera attribute
171         this.camera = camera;
172     }
173 }
174
175 }
176

```

Next, two local variable are inserted into the file above the instantiation of the player attribute. These are playerX and playerY and are included as arguments as the coordinates of the player when it is instantiated. They are assigned random values of coordinates in the maze. This is shown in the image below.

```

103     // Create a random position in the maze for the player
104     var playerX = Math.floor((Math.random() * this.maxX-1) + 1) * TILE_SIZE;
105     var playerY = Math.floor((Math.random() * this.maxY-1) + 1) * TILE_SIZE;
106
107
108
109     // Instantiate a new player attribute
110     // at the random position (playerX,playerY)
111     this.player = new Player(
112         new Sprite("hsl(145, 80%, 40%)",
113             playerX,
114             playerY,
115             [false,false,false,false]
116         )
117     );

```

Next, the rest of the local variables are added at the end of the class function definition – panning and fixedMap. These are simple Boolean variable that are assigned the values true and false respectively by default.

```

179
180    // Stores whether the camera is currently panning
181    // independent of player movement
182    var panning = true;
183
184    // Stores whether the level fits within the camera window and
185    // therefore disables the camera
186    var fixedMap = false;
187 }
188

```

Next, the initialise method was entered into the maze.js file. This is shown in the image below. This contains some temporary code that will be removed when the maze generation method has been written.

```

188 this.initialise = function() {
189
190     // ... this will contain code that generates a random maze
191     // ... for now just populate an empty maze
192
193     // Iterate from zero to max tile size in x direction
194     for (var i = 0; i < this.maxX; i++) {
195         // Instantiate a nested array of length maxY inside tiles attribute
196         this.tiles[i] = new Array(this.maxY);
197
198         // Iterate from zero to max tile size in y direction
199         for (var j = 0; j < this.maxY; j++) {
200             // Instantiate new sprite at ith,jth index
201             this.tiles[i][j] = new Sprite(
202                 this.randomFloor(),
203                 i*TILE_SIZE,
204                 j*TILE_SIZE,
205                 [false,false,false,false]
206             );
207         }
208     }
209     // ... end of temporary code
210 }
211

```

The position of the exit tile is then algorithmically determined using a heuristic method. The system will attempt to make a guess at a random position in the maze for the exit tile to be placed. If that coordinate in either horizontal or vertical axes is greater than the absolute distance of a third of the size of the maze then another guess at a suitable coordinate is made. This repeats until a suitable coordinate is found. The exitTile attribute is then instantiated at this position. This is shown in the image below.

```

205
210    // Select an acceptable exit zone
211    // randomExitX and randomExitY represent random coordinates
212    // acceptable being true represents an acceptable set of coordinates
213    var randomExitX = 0,
214        randomExitY = 0,
215        acceptable = false;
216
217    // Repeat until an acceptable value is found
218    while (!acceptable) {
219        // Generate a random value
220        randomExitX = Math.floor((Math.random() * this.maxX-1) + 1);
221        randomExitY = Math.floor((Math.random() * this.maxY-1) + 1);
222
223        // If the absolute distance between player and exit is third of map, values are acceptable
224        if (Math.abs(randomExitX - (this.player.x/TILE_SIZE)) > this.maxX/3 &&
225            Math.abs(randomExitY - (this.player.y/TILE_SIZE)) > this.maxY/3) acceptable = true;
226    }
227
228    // Instantiate the exitTile attribute at the coordinates that were generated above
229    this.exitTile = new ExitTile(
230        new Sprite(
231            EXIT_COLOUR,
232            randomExitX*TILE_SIZE,
233            randomExitY*TILE_SIZE,
234            [false,false,false,false]
235        );
236    );

```

Next, in a double nested for loop each of the tiles in the maze is iterated through with a pair of indexes x and y. Each of the tiles in the maze is pushed to the spriteBatch attribute's batch attribute. Then, the exitTile and player's sprites are pushed to the batch of spriteBatch. This is so that they are rendered on top of the other tiles.

```

237    );
238
239    // Push all of the cell sprites to the batch
240    // This is so they can be drawn to canvas
241    for (var x = 0; x < this.maxX; x++) {
242        for (var y = 0; y < this.maxY; y++) {
243            this.spriteBatch.batch.push(this.tiles[x][y]);
244        }
245    }
246
247    // Push the player and exit tile last
248    // They will therefore be drawn on top of tiles
249    this.spriteBatch.batch.push(this.exitTile.sprite);
250    this.spriteBatch.batch.push(this.player.sprite);
251
252}
253

```

An if statement then checks whether the size, in pixels, of the maze is less than that of the dimension's of the camera. If it is then panning is set to false and fixedMap is set to true. The camera is then centred on the maze in the x and y coordinates. This means that the camera will be static for the duration of this maze. The else statement starts by positioning the coordinate's of the camera on the centre of the exit tile. This is necessary so that when the maze begins the maze can pan from the exit tile to the player (and as such give them a view of the maze they need to solve). This is shown in the image below. This concludes the initialise method.

```

251
252    // Determine if the grid wholly fits in the viewport
253    if (this.maxX * TILE_SIZE <= this.camera.width &&
254        this.maxY * TILE_SIZE <= this.camera.height) {
255        // If it does, declare the map fixed and centre the grid on the viewport
256        panning = false; fixedMap = true;
257
258        // Centre the camera on the maze
259        this.camera.x = (this.maxX*TILE_SIZE/2) - (this.camera.width/2);
260        this.camera.y = (this.maxY*TILE_SIZE/2) - (this.camera.height/2);
261
262    } else {
263        // Start the camera looking at the exit zone
264        // (Camera will then pan to player in update loop)
265        this.camera.x = (randomExitX*TILE_SIZE * 2 + TILE_SIZE)/2 - (this.camera.width/2);
266        this.camera.y = (randomExitY*TILE_SIZE * 2 + TILE_SIZE)/2 - (this.camera.height/2);
267    }
268
269}
270

```

A draw method is written which simply calls the draw method of the spriteBatch attribute. An update method is written and left empty for the time being. A randomFloor method is written that

generated a random shade of grey and concatenates the value into a string that can be used by the canvas. This is the return value of this particular method. These three methods are shown in the image below.

```

271  // Method draws all of the sprites to the canvas
272  this.draw = function() {
273      // Call draw on spriteBatch attribute
274      // Draws all the sprites in the batch
275      this.spriteBatch.draw();
276  }
277
278  // Method updates all of the maze state and data
279  this.update = function() {
280      // ... Add functionality here later.
281  }
282
283  // Methods generates a random shade of grey
284  this.randomFloor = function() {
285      // Generate random numerical rgb value
286      var g = Math.floor((Math.random() * 255) + 1);
287      // Concatenate into a colour string and return
288      return "rgb(" + g + "," + g + "," + g + ")";
289  }
290
291 }
```

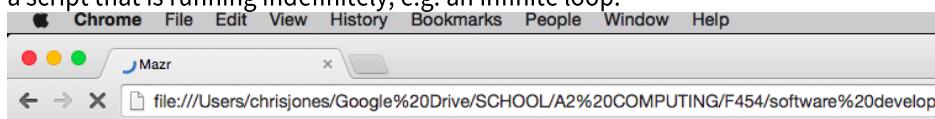
### First alpha test of Maze

The first alpha testing stage for the Maze.js file begins by seeing if that a maze object can be initialised and rudimentary drawing can be completed. As such, some test code is written that will be able to draw a simple maze to canvas. This was entered at the bottom of the maze.js file and will be removed at the end of the alpha testing. This test code is shown in the image below. Note the lack of comments is due to the code only being for alpha testing.

```

296
297  const TILE_SIZE = 32;
298  const BACKGROUND_COLOUR = "#333333";    // The colour of the background in hex
299  const WALL_COLOUR = "#222222";        // The colour of the walls in hex
300  const EXIT_COLOUR = "#ff8000";        // Colour of the exit in rgb
301
302  // Get the canvas drawing context
303  var canvas = document.getElementById('canvas');
304
305  canvas.width = 512;
306  canvas.height = 512;
307
308  var canvasContext = canvas.getContext('2d');
309
310  var maze = new Maze(8,8, canvasContext, new Camera());
311  maze.initialise();
312
313  maze.draw();
314 }
```

The index.html file was opened locally in the Google Chrome web browser. However, the page did not load. The image below shows that the tab bar hung with an indefinite ‘loading’ symbol and no webpage rendering occurred. I recognised this as an error that occurs when JavaScript is executing a script that is running indefinitely, e.g. an infinite loop.



In order to remedy this error, I realised that the heuristic placement of the exit tile was not finding the correct location because the player’s coordinates were NaN (Not a Number) at this point. This resulted in an infinite while loop that was never terminated and resulted in the ‘hanging’ loading icon and no rendering. I remedied this problem by moving the instantiation of the spriteBatch and player attributes and the declaration of player and playerY local variables to beneath the rest of the instantiation code. This would mean that the player’s coordinates are therefore defined after maxX and maxY are instantiated and thus do not end up being NaN.

```
170 // The sprite batch that will draw the grid
171 this.spriteBatch = new SpriteBatch(canvasContext, this.camera);
172
173 // Create a random position in the maze for the player
174 var playerX = Math.floor((Math.random() * this.maxX-1) + 1) * TILE_SIZE;
175 var playerY = Math.floor((Math.random() * this.maxY-1) + 1) * TILE_SIZE;
176
177 // Instantiate a new player attribute
178 // at the random position (playerX,playerY)
179 this.player = new Player(
180     new Sprite("hsl(145, 80%, 40%)",
181             playerX,
182             playerY,
183             [false,false,false,false]
184         )
185     );
186
187 );
```

The image below shows an excerpt of webpage that shows the rendering of a maze on a canvas. Note the presence of the orange exit tile and the green player tile. The rest of the tiles are also shades of black. This is correct behaviour at this point, although the maze is currently completely static as no update or positioning code has been written. However, one potential issue with this is that the shades of grey are not ‘light grey’ as specified in **Design objective 3.l.viii**. The algorithm that achieved this was incorrectly translated from pseudo code to JavaScript.



To fix this, I edited the randomFloor method. Instead of generating an RGB colour value in the range 0-255, it would generate an RGB colour value in the range 205-255. Limiting the possible values of the random number to 0-50 and then subtracting this value from 255 achieved this. The string is concatenated and returned in the same fashion. The modified code is shown in the image below.

```
// Methods generates a random shade of grey
this.randomFloor = function() {
    // Generate random numerical rgb value
    var g = 255 - Math.floor((Math.random() * 50) + 1);
    // Concatenate into a colour string and return
    return "rgb(" + g + "," + g + "," + g + ")";
}
```

Below is an image of the result of the change when the index.html file is opened in a webpage. Note that the colours are much more pleasant and will look very good against the dark grey walls when they are included later in the section. This concludes the first alpha test for the maze.js file



### Further development of Maze class function

The update method that was written earlier but left empty will now be populated. An if statement polls the run method of the completionTimer to execute every second. Each second the completedSeconds attribute is incremented. Beneath this if clause, the colour of the player's sprite is assigned to the return value of a call to the player's fadeColour method. This is shown in the image below.

```

282
283     // Method updates all of the maze state and data
284     this.update = function() {
285         // Wait every 1 second (1000ms) for completionTimer to elapse
286         if (this.completionTimer.run(1000)) {
287             // Increment the second counter by 1 every second
288             this.completedSeconds+=1;
289         }
290
291         // Update the player's colour
292         // This colour changes over time so must be assigned on every update cycle
293         this.player.sprite.colour = this.player.fadeColour();
294     }

```

Beneath this, code is written that controls whether the camera will pan at the beginning of a game. If the panning variable is set to false, animatePosition method is called every 10ms by nesting it in an if statement polling the run method of animationTimer. Beneath this, if the map is not fixed, then the camera is updated so that its coordinates match the centre of the player – effectively causing the camera to follow the player around the maze. If the camera is panning, then the panningTimer is polled every 10ms. The contents of this if clause is left empty for the time being. This code is shown in the image below.

```

294
295     // If the camera is not currently panning from exit to player
296     if (!panning) {
297         // Execute every 0.01 seconds (10ms)
298         if(this.animationTimer.run(10)) {
299             // Animate sprite's position movement
300             this.player.animatePosition();
301         }
302         // Clamp the centre of the camera viewport to the player
303         // If the maze isn't small enough to be fixed to the centre
304         if (!fixedMap) {
305             // Set x and y coordinates of camera to centre of player
306             this.camera.x = (2*this.player.x + TILE_SIZE)/2 - (WIDTH/2);
307             this.camera.y = (2*this.player.y + TILE_SIZE)/2 - (HEIGHT/2);
308         }
309         // Else execute this if camera is panning
310     } else {
311         // Execute every 0.01 seconds (10ms)
312         if (this.panningTimer.run(10)) {
313             // DO CAMERA PANNING HERE...
314         }
315     }
316 }
317 }
318

```

The code can now be added that is inside the if clause than polls the run method of panningTimer. The purpose of the code in the [image below](#) is to increase by a small proportion (an eighth of the tile size in the maze) the position of the camera in the x direction so that it is closer to the player's x coordinate. This involves an if-else statement, as the direction in which the camera moves in the x-axis in order to move closer to the player must be selected. The first if statement checks whether the x coordinate of the camera is smaller than the x coordinate of the player, adjusted with respect to the camera. If it is then the camera's x coordinate is incremented by TILE\_SIZE divided by 8. Otherwise if the x coordinate of the camera is greater than the player, adjusted with respect to the camera, then the x coordinate of the camera is decremented by TILE\_SIZE divided by 8.

```

312         // Execute every 8ms seconds (20ms)
313         if (this.panningTimer.run(10)) {
314             // Update the x pos of the camera to pan towards player
315             // If the camera has a lower x coordinate than the player
316             // Move the camera right
317             if (this.camera.x < (this.player.x *2 + TILE_SIZE)/2 - (WIDTH/2)) {
318                 // Move eighth of width of tile (4px) towards the player in positive x
319                 this.camera.x += TILE_SIZE/8;
320             // If the camera has a greater x coordinate than the player
321             // Move the camera left
322             } else if (this.camera.x > (this.player.x *2 + TILE_SIZE)/2 - (WIDTH/2)) {
323                 this.camera.x -= TILE_SIZE/8;
324             }
325         }
326     }

```

This code is repeated except now the y coordinate of the camera and player and substituted. This is shown in the image below.

```

324
325         // Update the y pos of the camera to pan towards player
326         // If the camera has a lower y coordinate than the player
327         // Move the camera down
328         if (this.camera.y < (this.player.y + this.player.y + TILE_SIZE)/2 - (HEIGHT/2)) {
329             this.camera.y+= TILE_SIZE/8;
330         // If the camera has a greater y coordinate than the player
331         // Move the camera up
332         } else if (this.camera.y > (this.player.y + this.player.y + TILE_SIZE)/2 - (HEIGHT/2)) {
333             this.camera.y-= TILE_SIZE/8;
334         }
335

```

The next if statement checks whether the coordinates of the camera equal the coordinates of the player with respect to the camera. If they do then panning can be considered done (as the camera has panned over to the player and now they share the same coordinates) so panning is set to false. This is shown in the image below.

```

335
336         // Check when the camera is centered on the sprite then cease to pan
337         // Compare position of camera to that of player
338         if (this.camera.y == (this.player.y + this.player.y + TILE_SIZE)/2 - (HEIGHT/2) &&
339             this.camera.x == (this.player.x + this.player.x + TILE_SIZE)/2 - (WIDTH/2) ) {
340             // Cease to pan when positions are the same
341             // by setting panning to false
342             panning = false;
343         }
344     }
345 }
346
347 }

```

Inside the main body of the update method, An if clause checks whether the x and y coordinates of the player are equal to the x and y coordinates of the exit tile (the player and exit tile would thus be in the same position). If they are then the complete attribute is set to true. This is shown in the image below.

```

347         // Check if the game has been completed
348         // Compare position of player with the exit tile
349         if (this.player.x == this.exitTile.x &&
350             this.player.y == this.exitTile.y) {
351             // If they are equal treat the maze as completed
352             // Set complete to true
353             this.complete = true;
354         }
355     }

```

The updatePlayerPosition method is now entered into the Maze class function. This method is left empty for the time being as it is dependent on making numerous calls to another method, colourTrail(), which will be written next. This method selects the tile underneath the player and colours it with the correct accent colouring. colourTrail() contains two local variables playerXPosition and playerYPosition, which represent the positions of the player tile in terms of indexes in the tiles array rather than absolute pixel coordinates. This is achieved by dividing the respective pixel coordinates of the player by TILE\_SIZE, which yields a result that can be used as an index. As such, the index operator is applied to the tiles attribute array to retrieve the tile sprite that

occupies the same position as the player. The colour attribute of this sprite is then set to that of the return value of a call to the trailColour method of the player. This is shown in the image below.

```

364
365    // Updates the position of the player within the maze
366    this.updatePlayerPosition = function(direction) {
367
368    }
369
370    // This method colours the tile beneath the player with the trail colour
371    this.colourTrail = function() {
372        // Get x and y position of the player
373        // Divide both the tile size so it can be used as an array index
374        var playerXPosition = this.player.x / TILE_SIZE;
375        var playerYPosition = this.player.y / TILE_SIZE;
376
377        // Use two previously calculated values to set colour attribute of
378        // the sprite beneath player to return value of player's trailColour
379        // function
380        this.tiles[playerXPosition][playerYPosition].colour =
381            this.player.trailColour();
382    }
383
384

```

With colourTrail() written, the updatePayerPosition() method will now be written. This method takes one parameter, direction, which is a string that specifies the direction the player is to move in by one tile (e.g. “up” will move the player up by one tile, etc.) The method initially contains an if-else statement that accepts strings for “up”, “down”, “left” and “right”. A final else statement triggers if a different string is passed as direction and an appropriate error is thrown. This is shown in the image below.

```

364
365    // Updates the position of the player within the maze
366    this.updatePlayerPosition = function(direction) {
367        if (direction == "up") {
368
369        } else if (direction == "down") {
370
371        } else if (direction == "left") {
372
373        } else if (direction == "right") {
374
375        } else {
376            // Invalid input
377            // Raise invalid argument error here
378            throw "Argument error. Direction specified must be 'up', 'down', 'left', 'right'";
379        }
380
381

```

The contents of the if statement that returns true if direction equals “up” was written next. A nested if statement checks if the newPositionX and newPositionY properties equal the x and y properties of the player and that the player is not disabled. These values are only equal when the player is not currently moving and thus is static on a tile. The purpose of this is to ensure the player is not moving before another movement is applied in this method. Next, the colourTrail() method is called so that the tile under the player is coloured with the appropriate accent. Then, another nested if statement checks if any walls block the path of the player, by checking if the top wall of the tile the player is currently on and the bottom wall of the tile the player is currently beneath are set to false. If this is true, then the newPositionY attribute of the player is set to be one TILE\_SIZE constant less than the current y coordinate. The stepsTaken attribute is also incremented by one so that a record of the number of steps the player makes is recorded for each movement. This is shown in the image below.

```

252     if (direction == "up") { // Move player up
253         // Check the player isn't moving by comparing new position to
254         // current position and that player is not disabled
255         if ((this.player.newPositionY == this.player.y &&
256             this.player.newPositionX == this.player.x) &&
257             !this.player.disabled) {
258
259             // Colour in the tile player is on
260             this.colourTrail();
261
262             // Check there are no walls blocking the path of the player
263             if (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[0] == false &&
264                 this.tiles[this.player.x/TILE_SIZE][(this.player.y-TILE_SIZE)/TILE_SIZE].walls[2] == false) {
265
266                 // Update the player's new y position
267                 // Subtract TILE_SIZE from y property of player
268                 this.player.newPositionY = this.player.y - TILE_SIZE;
269
270                 // Increment the number of steps taken
271                 this.stepsTaken += 1;
272
273             }
274         }

```

In the else-if clause beneath the first if clause, the “down” direction is now handled. This is completed in almost an identical fashion to that of above, except that the third nested if statement on line 400 checks whether bottom wall of the tile occupied by the player and the top wall of the tile below the player are set to false. If this is true, then the newPositionY attribute of the player is set to be one TILE\_SIZE constant more than the current y coordinate. The stepsTaken attribute is also incremented by one. This is shown in the image below.

```

390     } else if (direction == "down") {
391         // Check the player isn't moving and isn't at bottom of map or disabled
392         if ((this.player.newPositionY == this.player.y &&
393             this.player.newPositionX == this.player.x) &&
394             !this.player.disabled) {
395
396             // Colour in the square the player is on
397             this.colourTrail();
398
399             // Check there are no walls blocking the path of the player
400             if (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[2] == false &&
401                 this.tiles[this.player.x/TILE_SIZE][(this.player.y+TILE_SIZE)/TILE_SIZE].walls[0] == false) {
402
403                 // Update the player's new y position
404                 this.player.newPositionY = this.player.y + TILE_SIZE;
405                 this.stepsTaken += 1;
406
407             }
408
409         }

```

The next else-if clause again handles the same functionality except now that it deals with the “left” direction”. This is achieved by ensuring the third nested if statement, on line 419, checks that the left wall of the tile that the player currently occupies and the right wall of the tile to the left of the player are set to false. If this is true, then the newPositionX attribute of the player is set to be one TILE\_SIZE constant less than the current x coordinate. The stepsTaken attribute is also incremented by one. This is shown in the image below.

```

409▼     } else if (direction == "left") {
410         // Check the player isn't moving and isn't at leftmost y tiles of map or disabled
411         if ((this.player.newPositionY == this.player.y &&
412             this.player.newPositionX == this.player.x) &&
413             !this.player.disabled) {
414
415             // Colour the square the player is on
416             this.colourTrail();
417
418             // Check there are no walls blocking the path of the player
419             if (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[3] == false &&
420                 this.tiles[(this.player.x-TILE_SIZE)/TILE_SIZE][this.player.y/TILE_SIZE].walls[1] == false) {
421
422                 // Update the player's new x position
423                 this.player.newPositionX = this.player.x - TILE_SIZE;
424                 this.stepsTaken += 1;
425
426             }
427
428         }

```

Finally, the else-if clause for the “right” direction is written. This again involves the same procedure as for the other directions. However, in the third nested if statement on line 339, the if statement checks if the right wall of the tile the player currently occupies and the left wall of the tile to the right of the player are set to false. The next else-if clause again handles the same functionality

except now that it deals with the “left direction”. This is achieved by ensuring the third nested if statement, on line 419, checks that the left wall of the tile that the player currently occupies and the right wall of the tile to the left of the player are set to false. If this is true, then the newPositionX attribute of the player is set to be one TILE\_SIZE constant more than the current x coordinate. The stepsTaken attribute is also incremented by one. This is shown in the image below.

```

428
429 } else if (direction == "right") {
430     // Check the player isn't moving and isn't at rightmost y tiles of map or disabled
431     if ((this.player.newPositionY == this.player.y &&
432         this.player.newPositionX == this.player.x) &&
433         !this.player.disabled) {
434
435         // Colour the square the player is on
436         this.colourTrail();
437
438         // Check there are no walls blocking the path of the player
439         if (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[1] == false &&
440             this.tiles[(this.player.x+TILE_SIZE)/TILE_SIZE][this.player.y/TILE_SIZE].walls[3] == false) {
441
442             // Update the player's new x position
443             this.player.newPositionX = this.player.x + TILE_SIZE;
444             this.stepsTaken += 1;
445
446         }
447     }

```

Next, the newMaze method must be written, which generates a two dimensional array that can be used to produce the correct walls of the maze (by modifying the temporary code that was entered into the initialise method earlier). In order to complete this code the maze generation algorithm research undertaken will be used in order to best solve the problem, including using the online tutorial on generating mazes in JavaScript (see this section for more information).

A newMaze method is created and inside a number of variables are instantiated. ‘tiles’ is an array that will store the configuration of the maze, whereas unvisitedTiles will store Boolean values for all of the tiles in the maze which helps to determine whether they have been visited or not. A double nested for loop is used to initialise both these arrays with their default values – [0,0,0,0] for tiles and true for unvisitedTiles. This is shown in the image below.

```

469     // Method generates a new maze
470     this.newMaze = function() {
471         // Declare tiles as empty array
472         var tiles = [];
473
474         // Declare unvisitedTiles tiles as empty array
475         var unvisitedTiles = [];
476
477         // Iterate through y tiles up to maxY with counter y
478         for (var x = 0; x < this.maxX; x++) {
479             // Set each index in tiles to empty arrays
480             tiles[x] = [];
481             // Set each index in unvisitedTiles to empty arrays
482             unvisitedTiles[x] = [];
483
484             // Iterate through x tiles up to maxX with counter x
485             for (var y = 0; y < this.maxY; y++) {
486                 // Set the tile at index i,j to a 4 element array of zeroes
487                 tiles[x][y] = [0,0,0,0];
488                 // Set the same index in unvisitedTiles to true
489                 unvisitedTiles[x][y] = true;
490             }
491         }
492     }

```

Next, two variables randomX and randomY are declared as a pair of random coordinates in the maze. These are then inserted as the two elements of currentTile. A tileStack variable is then declared which contains at the moment a single element, currentTile. Then, the unvisitedTile at the coordinates of the currentTile variable is set to false (as it is counted as visited). visitedTiles is a counter that keeps a track of the number of tiles that have been visited thus far in the maze. It is set to 1 as the currentTile is considered the first visited tile. This is shown in the image below.

```

493     // Determine random x and y positions in the maze
494     // Constrain by values maxX and maxY
495     var randomX = Math.floor(Math.random()*this.maxY);
496     var randomY = Math.floor(Math.random()*this.maxX);
497     // Declare current cell as array with elements randomX and randomY
498     var currentTile = [randomX, randomY];
499
500     // Declre path as array with one element - currentTile
501     var tileStack = [currentTile];
502
503     // Set the current tile as visited
504     unvisitedTiles[currentTile[0]][currentTile[1]] = false;
505
506     // Set visitedTiles equal to 1
507     var visitedTiles = 1;
508

```

A while loop is then written which will be used to iteratively carve out a path in the maze. While the number of tiles visited is less than the total number of tiles in the maze (the condition of the loop), the code will execute. Within the for loop, potentialNeighbours is declared which contains an array of possible coordinates of tiles that are adjacent to the currentTile. This is necessary because all of the neighbours at this stage are potential and are inserted into the array potentialNeighbours. neighbourTiles will then be populated with the validated elements. of potentialNeighbours. An if statement is used to check if any of these elements are out of bounds or have already been visited. If the neighbour is in bounds and is unvisited then it is pushed onto neighbourTiles. This is shown in the image below.

```

508     // Loop through all available cell positions
509     while (visitedTiles < this.maxX * this.maxY) {
510         // Determine potentialNeighboursential neighboring tiles
511         var potentialNeighbours = [
512             [currentTile[0]-1, currentTile[1], 0, 2],
513             [currentTile[0], currentTile[1]+1, 1, 3],
514             [currentTile[0]+1, currentTile[1], 2, 0],
515             [currentTile[0], currentTile[1]-1, 3, 1]
516         ];
517         var neighbourTiles = [];
518
519         // Determine if each neighboring cell is in game grid, and whether it has already been checked
520         for (var l = 0; l < 4; l++) {
521             if (potentialNeighbours[l][0] > -1 &&
522                 potentialNeighbours[l][0] < this.maxY &&
523                 potentialNeighbours[l][1] > -1 &&
524                 potentialNeighbours[l][1] < this.maxX &&
525                 unvisitedTiles[potentialNeighbours[l][0]][potentialNeighbours[l][1]]) {
526                 neighbourTiles.push(potentialNeighbours[l]);
527             }
528         }
529     }
530

```

Next, an if statement checks if there is at least one element in neighbourTiles. This is essentially checking if one suitable neighbour has been found. selectedTile is declared as a randomly selected name from neighbourTiles. Then, the wall between currentCell and selectedCell is removed. This in fact involves two removals, the removal of one wall from currentCell and the removal of the adjacent wall from SelectedTile. Then, the tile at the coordinates of selectedTile is set as true in unvisitedTiles, as it has been visited. visitedTiles is incremented as this tile is counted as visited and the currentTile is set to selectedTile (this is in essence following the path). currentTile is then pushed onto the tileStack so that if backtracking is required it can be retrieved.

In the else clause, if no neighbours are found, then the currentTile is set to equal the top value of the tileStack (which is removed in the process). This allows for the process of backtracking to occur, as elements already visited will be continually popped from the tileStack until a tile with at least one suitable unvisited neighbour is found. This will continue until every tile in the maze is visited.

Finally, the tiles variable is returned so the caller of the method can leverage the generated maze. This is shown in the image below.

```

531     // If at least one active neighboring cell has been found
532     if (neighbourTiles.length >= 1) {
533         // Choose one of the neighbourTiles at random
534         selectedTile = neighbourTiles[Math.floor(Math.random()*neighbourTiles.length)];
535
536         // Remove the wall between the current cell and the chosen neighboring cell
537         tiles[currentTile[0]][currentTile[1]][selectedTile[2]] = 1;
538         tiles[selectedTile[0]][selectedTile[1]][selectedTile[3]] = 1;
539
540         // Mark the neighbor as visited, and set it as the current cell
541         unvisitedTiles[selectedTile[0]][selectedTile[1]] = false;
542
543         visitedTiles+=1
544         currentTile = [selectedTile[0], selectedTile[1]];
545         tileStack.push(currentTile);
546     }
547     // Otherwise go back up a step and keep going
548     else {
549         currentTile = tileStack.pop();
550     }
551 }
552 return tiles;
553
554 }
```

Finally, a modification needed to be made to the initialise method of Maze.js. This was necessary in order to use the result of the newMaze method, which is called and stored in a variable maze. Then, in the nested for loop which instantiates sprites in a grid, the walls are assigned as the results of the two dimensional array produced by newMaze, where the index of each one is equal to the coordinate of the tile. As Booleans were returned by newMaze a comparison to check if it is false is used – as a false represents a wall and a true represents a pathway. This is shown in the image below.

```

188  this.initialise = function() {
189      // Generate a new random maze
190      var maze = this.newMaze();
191
192      // Iterate from zero to max tile size in x direction
193      for (var i = 0; i < this.maxX; i++) {
194          // Instantiate a nested array of length maxY inside tiles attribute
195          this.tiles[i] = new Array(this.maxY);
196
197          // Iterate from zero to max tile size in y direction
198          for (var j = 0; j < this.maxY; j++) {
199              // Instantiate new sprite at ith,jth index
200              this.tiles[i][j] = new Sprite(
201                  this.randomFloor(),
202                  i*TILE_SIZE,
203                  j*TILE_SIZE,
204                  [maze[i][j][3] == false,
205                  maze[i][j][2] == false,
206                  maze[i][j][1] == false,
207                  maze[i][j][0] == false
208              ]
209          );
210      }
211  }
212
213 }
```

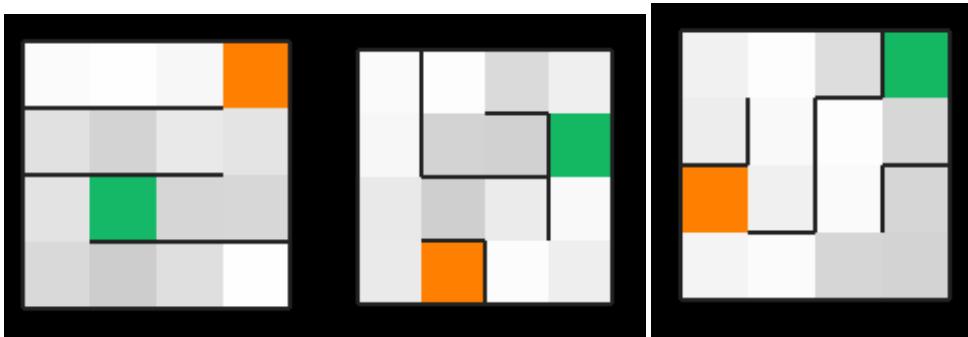
### Alpha testing the Maze generation

The test code used earlier was reinserted into the file and modified in order to test the random nature of the mazes and their ability to update. This is shown in the image below.

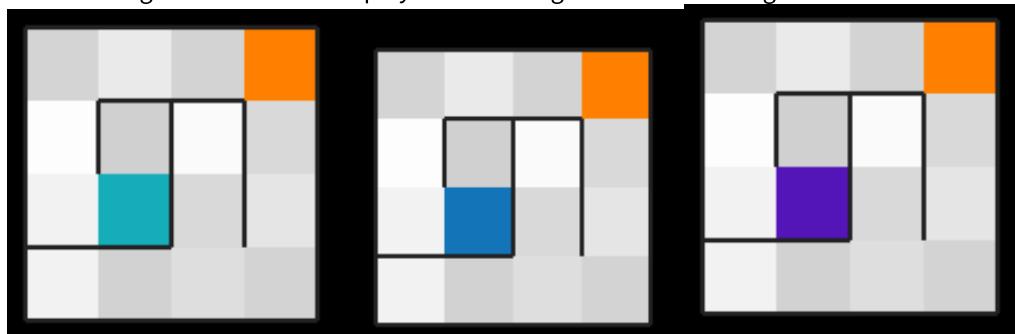
```

555 const TILE_SIZE = 32;
556 const BACKGROUND_COLOUR = "#333333";    // The colour of the background in hex
557 const WALL_COLOUR = "#222222";           // The colour of the walls in hex
558 const EXIT_COLOUR = "ff8000";   // Colour of the exit in rgb
559
560 const WIDTH = 512;
561 const HEIGHT = 512;
562
563 // Get the canvas drawing context
564 var canvas = document.getElementById('canvas');
565
566 canvas.width = 512;
567 canvas.height = 512;
568
569 var canvasContext = canvas.getContext('2d');
570
571 // set the canvas black
572 canvasContext.fillStyle = "black";
573 canvasContext.fillRect(0,0,512,512);
574
575 var maze = new Maze(4,4, canvasContext, new Camera());
576 maze.initialise();
577
578 setInterval(function() { // Anonymous function ...
579     maze.update();
580     maze.draw();
581 }, 1000/30);
582
583 maze.newMaze();
584
585
586 }
```

The webpage was visited in order to begin alpha testing. As shown in the images below, walls are being displayed that show that random mazes are being produced. These do appear random as upon a browser refresh being undertaken a new configuration is presented.



Note that the colour of the player also fades through a hue spectrum. This is desirable behaviour and the images below show the player has undergone a colour change.



Finally, testing the updatePlayerPosition method is necessary. To do this, the console was used to enter some movement position commands. These resulted in movement of the player (and the player leaving a coloured trail) as shown in the image below on the left. The console commands are included in the image on the right.

 A screenshot of a 4x4 grid-based maze with a colored trail. The trail consists of a sequence of colored squares (orange, blue, cyan) connected by black outlines, showing the path of the player. To the right of the maze is a terminal window showing the following command history:
 

```
> maze.updatePlayerPosition("up")
< undefined
> maze.updatePlayerPosition("right")
< undefined
> maze.updatePlayerPosition("up")
< undefined
> maze.updatePlayerPosition("left")
< undefined
> maze.updatePlayerPosition("left")
< undefined
> maze.updatePlayerPosition("down")
< undefined
>
```

This is satisfactory alpha testing at this stage, as thorough testing will be completed in the Testing section. Therefore, the engine.js file will now be written to finish developing the new system.

### Engine.js file creation

This section pertains to the development of Input Requirements 1, 2, 3, 4, 5, 6, 7, 8 and 9 and Processing Requirements 14, 15, 16 and 17 and Output Requirements 26 and 27.

The engine.js is one of the most important files in the development of the game as it is where the global scope is used in order to control the ‘pages’ of the page the player visits.

Engine.js is not itself a file that contains a class function so only a short multiline comment header is included. It is comprised of multiple sections labelled with more detailed comments for each one. The first section of code includes the global constants that are included in the global scope table of the data structures section of the nature of the solution. They are all declared with literal values and represent immutable data that will not change throughout the course of the game.

```

1 ▼ /*
2   engine.js
3   Global scope file
4   -----
5   Written on the 04/01/2014 by Chris Jones
6 */
7
8 // GLOBAL CONSTANTS
9 // These are immutable and do not change over course of program
10 //
11 const WIDTH    = 512; // Width of the viewport
12 const HEIGHT   = 512; // Height of the viewport
13 const TILE_SIZE = 32; // The size of the tiles
14
15 const FPS = 30; // The fixed FPS (frames per second)
16 // Controls the number of times screen is redrawn per second
17
18▼ const STATE = { // This is an object literal used as an 'enumeration'
19   MENU: 0, // This is the state of the game
20   MAZE: 1 // Used when it is necessary to check the current page
21 };
22
23 const BACKGROUND_COLOUR = "#333333"; // The colour of the background in hex
24 const WALL_COLOUR = "#222222"; // The colour of the walls in hex
25 const EXIT_COLOUR = "rgb(255,128,0)"; // Colour of the exit in rgb
26 //

```

Next, the global variables are initialised. An empty object literal GAME is declared into which some important methods will be inserted. The canvas is extracted from the HTML DOM (Document Object Model) and assigned to the variable canvas. The width and height properties of the canvas object are set to match the global constants WIDTH and HEIGHT. Then, the 2d drawing context of the canvas is stored in canvasContext. The height and width of the first maze are initialised, both as 4, and the level is initialised as 1. The variable maze stores the maze in which the game is played. This is instantiated as a new maze object width the mazeWidth and mazeHeight variables passed as arguments for width and height respectively. The canvasContext is also passed as the context onto which this maze will be drawn (by the Maze object’s SpriteBatch). A new camera is also instantiated with default values (as no arguments were passed) as the final argument to the Maze object. Finally, the variable currentState is declared and assigned the value of the MENU attribute of the STATE object. This is equal to a value of 0 and used to track whether the player is solving a maze or navigating a menu. Finally, the initialise method of the maze object is called in order to generate a new random maze.

```

29 // INstantiate GLOBAL VARIABLES
30 // These are implicitly defined to be global
31 // As they are in the global scope
32 //-----
33 // GAME is the object that wraps up the main functionality
34 // Methods and variables are added to this object later in the file
35 var GAME = {};
36
37 // Reference to the canvas element extracted from the HTML
38 var canvas = document.getElementById('canvas');
39
40 // Set the canvas dimensions equal to the constant dimensions
41 canvas.width = WIDTH;
42 canvas.height = HEIGHT;
43
44 // Reference to the drawing context of the canvas
45 var canvasContext = canvas.getContext('2d');
46
47 // The width of the maze initialised to 8
48 var mazeWidth = 4;
49
50 // The height of the maze initialised to 8
51 var mazeHeight = 4;
52
53 // Stores the level the player is on, starting at 1
54 var level = 1;
55
56 // Create a new maze object and store in variable maze
57▼ var maze = new Maze(
58   mazeWidth,
59   mazeHeight,
60   canvasContext,
61   new Camera()
62 );
63
64 var currentState = STATE.MENU;
65
66 // Initialise the newly instantiated maze
67 maze.initialise();
68 //-----
69

```

The next section of code deals with the navigation of the menus within the game. In accordance with the design specification the game and all menus will be rendered in a single web page and as such as menu and the game window will be rendered as ‘pages’ – HTML Divs (A HTML element that acts as a container) with content and styling to appear like individual pages within the main window. Only one page will be visible at any one time and a stack data structure is used to maintain the pages. Conveniently, JavaScript arrays have stack functionality built in.

Note the jQuery library will be used extensively in this section to manipulate the DOM with ease. This was included, as an external script in index.html and no other setup is required to begin using it. For more information on the library visit <http://jquery.com>.

Firstly, a variable pageStack is defined and assigned the value of an empty array. This will act as the stack of pages and store strings which refer to the CSS identifier of the Div in HTML.

Next, a small function showPage was written. This takes a single parameter paramString which is the CSS string selector required in order for jQuery to select that page.

Then, two calls are made to the jQuery object (represented by a \$ sign). The first calls the CSS method on all classes called “page” in the HTML and sets them to not display. This in essence hides all of the page divs on the webpage. Next, the CSS of the element with selector equal to paramString parameter are set to display (in inline-flex mode). This means that all pages except for that with selector paramString will be hidden on a call to showPage.

```
71 // NAVIGATION FUNCTIONS
72 //-
73 // Instantiate the page stack as an empty array
74 // This will act as a stack of pages visited by the user
75 var pageStack = [];
76
77 function showPage(pageString) {
78     // First hide all of the pages
79     $('.page').css('display', 'none');
80
81     // Then display the desired page
82     $(pageString).css('display', 'inline-flex');
83 }
84
```

The next function, defined with identifier navigateTo, takes the same parameter pageString as showPage.

An if statement is first written to check if it matches the class of the main-menu page div. If it does, another if statement nested in the first is used to check whether the level saved to the player's local storage is 1. If it is, the jQuery object is used and attr function is called to insert the disabled attribute into the HTML of the continue button (class 'btn-continue') The contents of the button's HTML is set to 'Continue', in case it contains a level message that had been inserted previously. Otherwise, the disable attribute is removed and some HTML is inserted into the contents of the button. This, when concatenated with the stored level, produces a message of which level the player continues from.

Another if statement is used to determine if the page at the top of the stack is equal to the element with ID 'canvas' (represented by the selector '#canvas'). If this returns true, then the player is disabled, then the canvas is being navigating away from so the player's movement is disabled and the currentState is set to equal MENU attribute of STATE. The next if state does effectively the opposite, checking if the player is navigating to the canvas and sets disabled to false if this is true and the current state to the MAZE attribute of STATE.

Outside of this if statement, the showPage function is called, passing the parameter pageString as an argument to the function. Then, pageString is pushed onto the pageStack so that it is at the top of the stack. This function is shown completed in the image below.

```

// Navigate to page with HTML id pageString
function navigateTo(pageString) {
    if (pageString == ".main-menu") {
        // If the currently stored level is one
        if (localStorage["level"] == 1) {
            // Disable the continue button
            $('.btn-continue').attr('disabled', 'disabled');
            // Set the text within the button to 'Continue'
            // This is incase it contains a level number
            $('.btn-continue').html('Continue');
        } else {
            // Otherwise insert the level the player is currently on
            $('.btn-continue').removeAttr('disabled');
            $('.btn-continue').html(
                'Continue <br><p class="under-text">From level ' +
                localStorage["level"] + "</p>"
            );
        }
    }

    // If navigating away from the canvas
    if (pageStack[pageStack.length-1] == "#canvas") {
        // Disable the player
        maze.player.disabled = true;
        // Switch the current state to menu
        currentState = STATE.MENU;
    }

    // If navigating to the canvas
    if (pageString == "#canvas") {
        // Enable the player
        maze.player.disabled = false;

        // Switch the current state to game
        currentState = STATE.MAZE;
    }

    // Show the page that is being navigated to
    showPage(pageString);

    // Push it to the top of the pageStack
    pageStack.push(pageString);
}

```

The final function in the navigation section is goBack(). This is a shortcut function in order to navigate the menus one page back. It takes no parameters.

Firstly, an if statement is implemented to check the length of pageStack is at least 2. This is because it makes no sense to attempt to move back one page. If this condition is met, an element is popped off the pageStack. This represents the top page being completely removed from the order of pages. Next, the navigateTo function is called with the argument being another call to the pop method of pageStack. While it may seem illogical to pop the off the page selector of the one we are navigating to, the navigateTo method will push it back onto the stack for us. This code is shown in the image below.

```

129 // Go back to the last visited page
130 ▼ function goBack() {
131     // Check there are at least two pages in the stack
132     // Otherwise there is no menu to navigate back to
133 ▼   if (pageStack.length >= 2) {
134     // Pop the top page off of the stack.
135     // This has the effect of navigating back a page
136     pageStack.pop();
137
138     // Navigate to the new top of the stack
139     // Note this is popped off here
140     // and pushed back on in navigation
141     navigateTo(pageStack.pop());
142   }
143 }
144 //-----
145

```

This concludes the section on page navigation functionality. Next, I begin to populate the GAME object. The first is to assign to it a method with the identifier ‘save’ and no parameters. Save uses the localStorage object and the index operator in order to maintain a hash table of stored key-value pairs. The keys are “level”, “x” and “y” for the current level of the player, the width of the maze and the height of the maze respectively. This is achieved by simply assigning the value for each key its respective variable.

```

147 // DEFINE THE FUNCTIONS / VARIABLES THAT CONTROL THE GAME
148 // These are bound to the object GAME
149 // This is instantiated above
150 //-----
151 // This function handles saving the state of the GAME to localStorage
152 GAME.save = function() {
153     // Store the value of level in key "level" of localStorage
154     localStorage["level"] = level;
155
156     // Store the value of mazeWidth in the key "x" of localStorage
157     localStorage["x"] = mazeWidth;
158
159     // Store the value of mazeHeight in the key "y" of localStorage
160     localStorage["y"] = mazeHeight;
161 }

```

The load function effectively does the inverse, retrieving from the localStorage object the width, height and level of the maze using the keys “x”, “y” and “level” respectively. Note the parseInt function is called into to convert the stored string values into integer values to be assigned correctly to the variables mazeWidth, mazeHeight and level.

```

162
163 // This function handles loading GAME state from localStorage
164 ▼ GAME.load = function() {
165     // Retrieve the width of the maze stored
166     mazeWidth = parseInt(localStorage["x"]);
167
168     // Retrieve the height of the maze store
169     mazeHeight = parseInt(localStorage["y"]);
170
171     // Retrieve the level that is stored
172     level = parseInt(localStorage["level"]);
173 }
174

```

The update function was then written. It calls the update method on the maze object, in essence updating the maze every time the game is updated. The jQuery object is used along with some string manipulation in order to produce a string of the time taken in the format ‘mm:ss’, which is

then placed in the HTML element with the id time. The element with id steps is also updated so that it contains the text of the number of steps taken. The method then uses an if statement to check if the maze is completed by polling the value of the maze object's complete attribute. This returns a Boolean so no logical comparison is necessary.

Within the if statement, a maze is reinstated with a new Maze object, with the mazeWidth and mazeHeight variables incremented by 2 and passed as arguments for width and height respectively. The maze initialise method is then called to reinitialise and generate a new maze. Next, the navigateTo method is called to navigate to the end of level menu page. Level is incremented by 1 and then the save method is called to save progress.

```
// Function handles all of the update logic necessary in the game
GAME.update = function() {
    // Update the logic of the maze
    maze.update();

    // Check if maze is complete
    if (maze.complete) {

        $('#time').html(
            (String(parseInt(maze.completedSeconds / 60)).length > 1 ? parseInt(maze.completedSeconds / 60) : "0" +
            parseInt(maze.completedSeconds / 60)) + ":" +
            (String(maze.completedSeconds).length > 1 ? maze.completedSeconds % 60 : "0" + maze.completedSeconds % 60));

        $('#steps').html(String(maze.stepsTaken));

        // If so, generate a new maze with two more cells in each axis
        maze = new Maze(mazeWidth+2, mazeHeight+2, canvasContext, new Camera());
        // Initialise this new maze
        maze.initialise();

        // Display the end of level menu
        navigateTo('.end-of-level-menu');

        // Increment level
        level +=1;

        // Save the value of the maze
        this.save();
    }
}
```

The draw function is a short function that I wrote next. It first renders the background by setting the canvasContext fillStyle property equal to the BACKGROUND\_COLOUR constant. The fillRect method is then call passing the dimensions of the canvas to the context. This fills the whole canvas with the background colour in order to clear the last rendered frame of the canvas. This is so it updates consistently.

Finally the draw method of the maze object is called in order to render the game.

```
198 // Draw the maze and associated sprites
199 GAME.draw = function() {
200
201     // Fill in the background colour
202     canvasContext.fillStyle = BACKGROUND_COLOUR;
203     canvasContext.fillRect(0,0,WIDTH,HEIGHT);
204
205     // Draw the maze to the canvas
206     maze.draw();
207 }
```

I wrote the doKeyDown function next which takes a single parameter e. This is general convention for a function that is added to an event listener – as this one will be; being added to the event listener detecting the 'keydown' event. A switch statement is used with the cases being tested against the keyCode property of e. This returns an integer value that represents the key on the keyboard being pressed. These can be found from a cheat sheet such as the one I found at <http://shikargar.wordpress.com/2010/10/27/javascript-key-codes/> (correct as of 04/01/2015, the image of which is reproduced below).

Esc 27	F1 112	F2 113	F3 114	F4 115	F5 116	F6 117	F7 118	F8 119	F9 120	F10 121	F11 122	F12 123	PrScr 44	ScrLk 145	Break 19
- ~ 192	! 2 50	@ 3 51	# 4 52	\$ 5 53	% 6 54	^ 7 55	& 8 56	*	( 9 57	) 0 48	- 189	= 187	Backspace 8		
Tab 9	Q 81	W 87	E 69	R 82	T 84	Y 89	U 85	I 73	O 79	P 80	{ [ 219	} ] 221	 \\ 220		
Caps Lock 20	A 65	S 83	D 68	F 70	G 71	H 72	J 74	K 75	L 76	:	" ; 222	Enter 13			
Shift 16	Z 90	X 88	C 67	V 86	B 66	N 78	M 77	< 188	> 190	?	/ 191	Shift 16			
Ctrl 17	Win 91	Alt 18			32					Alt 18	Win 92	Menu 93	Ctrl 17		

"Print Screen" has unknown consistency. Additionally, key does not fire keydown and keypress. The keyup event may or may not fire. Don't use.  
 Not consistent across the major browsers (IE, FF, Opera, Chrome, Safari). Values displayed are those of IE. Don't use.  
 Vendor keys (Windows only – not compatible with Mac, ...) Additionally, not all Windows only keyboards contain a right Windows key. Don't use.  
 Unknown consistency across browsers. Don't use.

The first four cases are given a pair of case values. This represents an arrow key one of the WASD keys that is used to control movement. Cases in JavaScript cascade down so having an empty case above a case will cause the functionality for both to be the same. For example, pressing either the up arrow key (38) or the "W" key (87) the same code is executed. This is repeated for all four of the movement directions in the code.

Next, the pause button must be handled. This is the final case and only has one condition to match – the value 27 (value of the escape key). There is an if statement that checks whether the top of the page stack is not on the pause menu already and the canvas is currently at the top of the stack of pages (pageStack). If it is, then the navigateTo function is called in order to navigate to the paused menu page. Otherwise, the else clause is executed and the goBack() function is called. This has the effect of returning to the game when on a pause menu

Home 36	End 35	PgUp 33	/ 111	*	106	- 109
NumLock 144						
7 Home 36/103	8 ↑ 38/104	9 PgUp 33/105				
4 ← 37/100	5 ↓ 12/101	6 → 39/102				107
1 End 35/87	2 ↓ 40/98	3 PgDn 34/99				Enter 13
← 37	↑ 40	→ 39				
0 Ins 45/96	.	Del 46/110				

```

209  // Event listener function that deals with input
210 ▼ GAME.doKeyDown = function(e) {
211     // Switch statement must on keyCode variable of the event
212 ▼   switch(e.keyCode) {
213       // Up arrow or W key codes
214       case 87:
215       case 38:
216           // Update player position by moving it up
217           maze.updatePlayerPosition("up");
218           break;
219       // Down arrow or S key codes
220       case 83:
221       case 40:
222           // Update player position by moving it down
223           maze.updatePlayerPosition("down");
224           break;
225
226       // Down arrow or S key codes
227       case 65:
228       case 37:
229           // Update player position by moving it left
230           maze.updatePlayerPosition("left");
231           break;
232
233       // Right arrow or D key codes
234       case 68:
235       case 39:
236           // Update player position by moving it right
237           maze.updatePlayerPosition("right");
238           break;
239
240       // Escape key code
241       case 27:
242           // If the current page is not the pause menu
243           // and the current page is the game canvas
244           ▼ if (pageStack[pageStack.length-1] != ".pause-menu" &&
245               pageStack[pageStack.length-1] == "#canvas") {
246
247               // Navigate to the pause menu
248               navigateTo('.pause-menu');
249
250           } else {
251
252               // Go back a step in the stack of pages
253               goBack();
254
255           }
256       }
257   }
258 //-----
259

```

The addEventListener method of the window object is called. The first parameter is a string representing the event that is being listened out for. The second is the function that will be executed when the event is activated and the final parameter is useCapture, which is an old legacy feature of browsers and is passed the value of false.

```

260
261 // INPUT HANDLER
262 //-----
263 // Add a keyboard event listener
264 window.addEventListener( "keydown", GAME.doKeyDown, false);
265 //-----
266

```

It was now time to begin the initialisation code of the main game loop. This code uses the window setInterval method in order to repeat the execution of a function a number of times with a delay of a set amount of seconds. It was used to test code in both the SpriteBatch and Player class function alpha testing sections. The setInterval method takes an anonymous function as the first argument (which is executed 30 times a second in this case). The second argument is 1000/FPS, the FPS being the constant 30. As the unit used by this method is milliseconds, 1000 ms is a second so we are effectively divided 1 second by 30, giving a frequency of game draw / update cycles of 30Hz.

Within the anonymous function there is an if statement in order to check that the game state is currently on a maze. This is in order to prevent it rendering mazes when the player is on the menu. Within this if statement, the GAME update and draw functions are called one after another in that respective order.

Finally outside of the setInterval menu the navigateTo menu is called with the appropriate argument to ensure the main menu is the first menu the player sees when they load the web page. This is shown written into the file in the image below.

```
279 // INITIALISE THE MAIN GAME
280 //-----
281 // setInterval is a function that takes a function as parameter.
282 // Takes time interval in milliseconds as second parameter
283 // Calls the function passed first every period that
284 // was passed as second parameter
285 setInterval(function() { // Anonymous function ...
286     // Check the game is in maze state before calling functions
287     if (currentState == STATE.MAZE) {
288         // Call the update method first
289         GAME.update();
290
291         // Call the draw method
292         GAME.draw();
293     }
294 }, 1000/FPS);
295
296 // Navigate to the main menu
297 navigateTo('.main-menu');
298 //-----
```

The following sections of code are jQuery events that represent the code that executes when different buttons are pressed in the game. The jQuery object is used by being passed a CSS selector and the click method means that I can attach a function to be executed on the occurrence of the respective button being clicked.

The first button, the continue button, has an anonymous function passed as an argument to the jQuery object. First the load method of the GAME object is called which loads game state data from local storage. Next, a new maze object is instantiated that implicitly uses the data sourced from local storage. The initialise method of maze is called. Finally the navigateTo function is called in order to navigate to the canvas page so that the player can begin solving a maze.

```

301 // EVENTS FOR BUTTON CLICKS
302 //-
303 // Continue button click event
304 $('.btn-continue').click(function() {
305     // Load the game state data from storage
306     GAME.load();
307
308     // Instantiate a new maze with respect to the loaded data
309     maze = new Maze(
310         mazeWidth,
311         mazeHeight,
312         canvasContext,
313         new Camera()
314     );
315
316     // Initialise the new maze
317     maze.initialise();
318
319     // Navigate to the canvas game page
320     navigateTo('#canvas');
321 });

```

The new game button, when clicked, sets the values of level, width and height to their respective defaults (1, 4 and 4). The maze is then instantiated with these new width and height dimensions passed as arguments to the new object. The initialise method of maze is called and the save method of GAME is called. This has the effect of overwriting the local storage of a game currently in play.

```

323 // New game button click event
324 ▼ $('.btn-new-game').click(function() {
325     // Set level to 1
326     level = 1;
327
328     // Set maze dimensions to 4
329     mazeWidth = 4;
330     mazeHeight = 4;
331
332     // Instantiate a new maze
333     maze = new Maze(
334         mazeWidth,
335         mazeHeight,
336         canvasContext,
337         new Camera()
338     );
339
340     // Initialise the new maze
341     maze.initialise();
342
343     // Save the game, at level one
344     // This ensures the continue button is enabled
345     GAME.save();
346
347     // Navigate to the game
348     navigateTo('#canvas');
349 });
350

```

The help button, when clicked, will simply call the navigateTo function and navigate to the first help page. This is shown in the image of the code below.

```
351 // Help button click event
352 $('.btn-help').click(function() {
353     // Navigate to the first help page
354     navigateTo('.help-1');
355});
```

The back button, when clicked, needs to make a decision based on the type of page the back is functioning from. An if statement checks if the top element of the pageStack contains the string 'help' and thus is a help page. If it is, a while loop is nested within this if statement which performs the same check. Within the while loop, there is a statement that pops off the top element of the pageStack (and discards it). This while loop has the effect of popping all the help pages off the top of the stack. When the loop finally reaches a page that doesn't contain the string help, then it will stop popping pages off the stack. Otherwise, if the top page isn't a help page, the goBack function is called. Below is an image of this function.

```
357 // Navigate back a page when back button is pressed
358▼ $('.btn-back').click(function() {
359    // Check if the top of the pageStack is a help page
360    // Back behavior is different for help pages
361▼   if (pageStack[pageStack.length-1].indexOf('help') > -1) {
362
363    // While there are still help pages in the stack
364▼     while (pageStack[pageStack.length-1].indexOf('help') > -1) {
365        // Pop this page off the stack
366        pageStack.pop();
367    }
368
369    // Navigate to top of the stack
370    navigateTo(pageStack.pop());
371
372    // Otherwise go back a page
373    } else {
374        goBack();
375    }
376});
```

The left button click event calculates which help page to navigate to. The first statement gets the last letter of the last element in pageStack and stores it in currentPageIndex. Next, the nextPage variable stores the parsed integer value of the string currentPageIndex, decremented by one. Finally, the navigateTo function is called, being passed a parameter of a concatenation of the string '.help-' and the nextPage variable. This has the effect of generating a string that references the last help page that the user wishes to visit. This is shown in the image below.

```

381 // Handle the left arrow button event
382 ▼ $('.left').click(function() {
383     // Get the index of the current help page
384     var currentPageIndex =
385         pageStack[pageStack.length-1][pageStack[pageStack.length-1].length-1];
386
387     // Subtract one from currentPageIndex
388     var nextPage = parseInt(currentPageIndex) - 1;
389
390     // Navigate to the previous help page in the sequence
391     navigateTo('.help-' + nextPage);
392 });
393

```

The right button click event has almost the exact same functionality as above, except that it increments the currentPageIndex by one instead of decrementing it by one. This is shown in the code below.

```

393
394 // Handle the right arrow button event
395 $('.right').click(function() {
396     // Get the index of the current help page
397     var currentPageIndex =
398         pageStack[pageStack.length-1][pageStack[pageStack.length-1].length-1];
399
400     // Add one to currentPageIndex
401     var nextPage = parseInt(currentPageIndex) + 1;
402
403     // Navigate to the next help page in the sequence
404     navigateTo('.help-' + nextPage);
405 });

```

When the next level button is clicked, the goBack function is called and executed. This is shown in the image below.

```

404 // Handle the next level button on the end of level menu
405 $('.btn-next-level').click(function() {
406     // Show the canvas game page
407     goBack();
408 });

```

The quit button click event is a simple anonymous function that contains a while loop that checks whether the length of pageStack is not equal to zero. If it is not, an element is popped off the top of pageStack. This has the effect of popping all the pages off the stack until it is empty. Finally, the save method of GAME is called in order to preserve game state. The navigateTo function is called passing the string parameter corresponding to the main menu.

```

410 // Handle the quit button event
411 $('.btn-quit').click(function() {
412     // Jump straight to the root menu
413     // Empty the whole page stack
414     // Do this by popping off all elements
415     // while the array still has elements
416     while (pageStack.length != 0) {
417         pageStack.pop();
418     }
419
420     // Save the game
421     GAME.save();
422
423     // Navigate to the main menu
424     navigateTo('.main-menu');
425 });
426 //-----

```

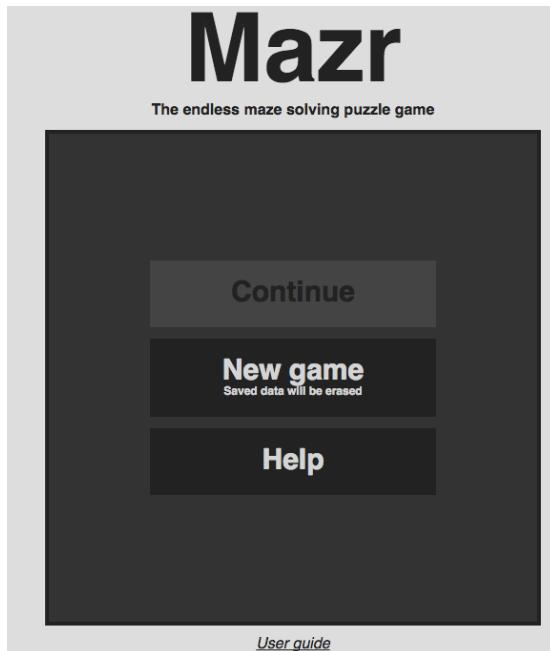
Finally, the stylesheet files need to be uncommented in index.html in order to begin alpha testing. This is shown in the image below.

```
8      <!-- Link to stylesheets -->
9      <link rel="stylesheet" href="styles/reset.css" />
10     <link rel="stylesheet" href="styles/stylesheet.css" />
11
```

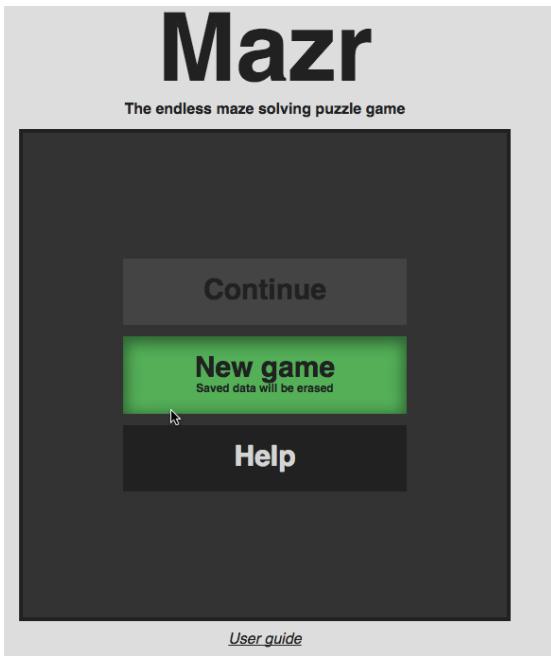
### Alpha Testing Engine.js

This alpha testing presents an overview of the whole of the operation of the game, as other than for modifications, the scripts have been written and should function properly. Although bugs are inevitable, some superficial ones may be corrected below or later in the Testing section.

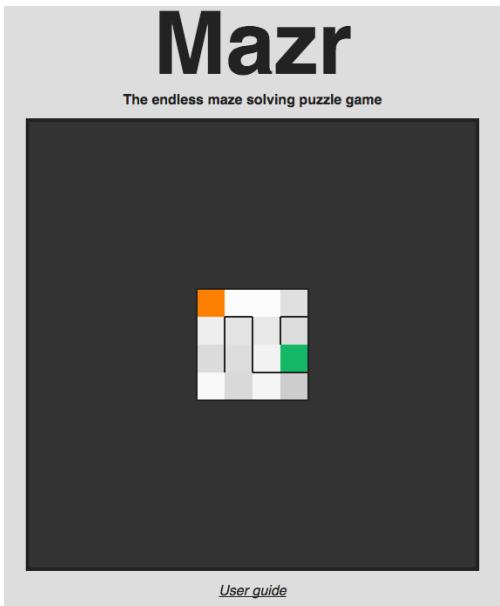
First, I opened index.html in Google Chrome. It was pleasing to see the styling and pages were rendering appropriately. Further, the game soundtrack was playing and looping which was a success.



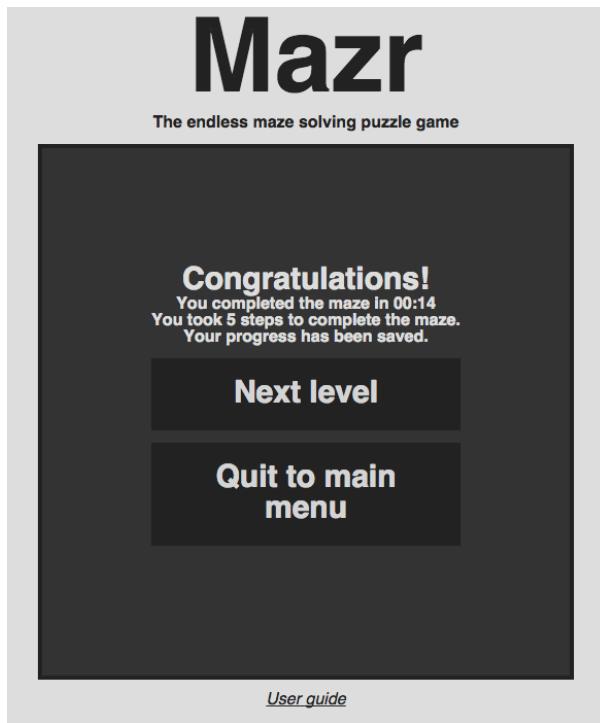
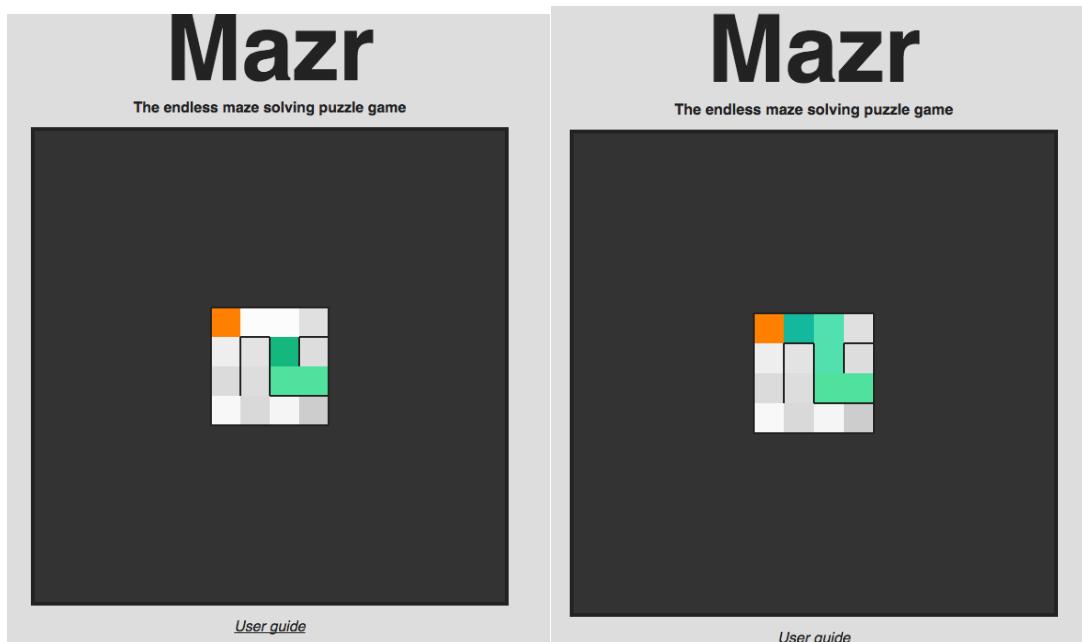
Next, I pressed and held the 'New game' button. The correct focus styling had been applied and is shown below.



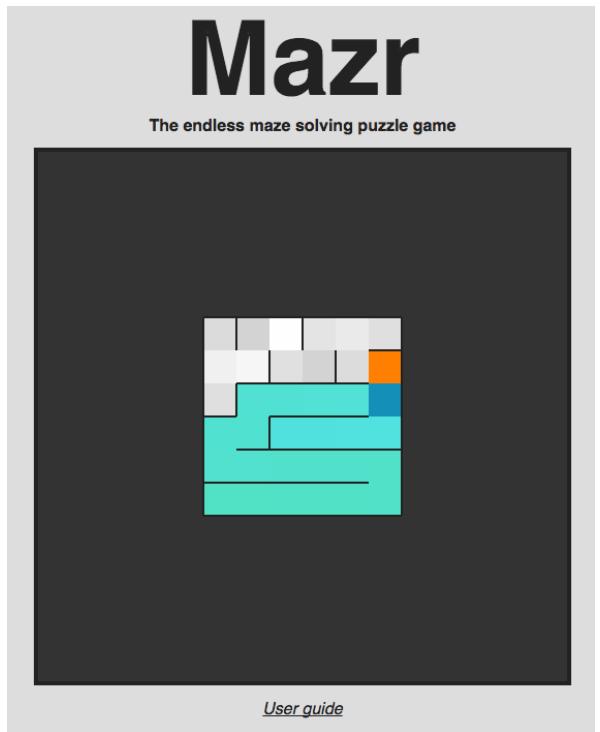
Upon releasing the new game button, the first maze loaded. The tiles in the maze, after roughly counting, amounted to a total of 16 that was correct for a 4x4 maze on level 1. An image of this maze in the game is shown below. Note the presence of a start green tile and orange exit tile along with a perfect maze (in which any square is reachable from any other square). Everything appeared to be drawing correctly, including the maze in being at the centre of the view of the camera.



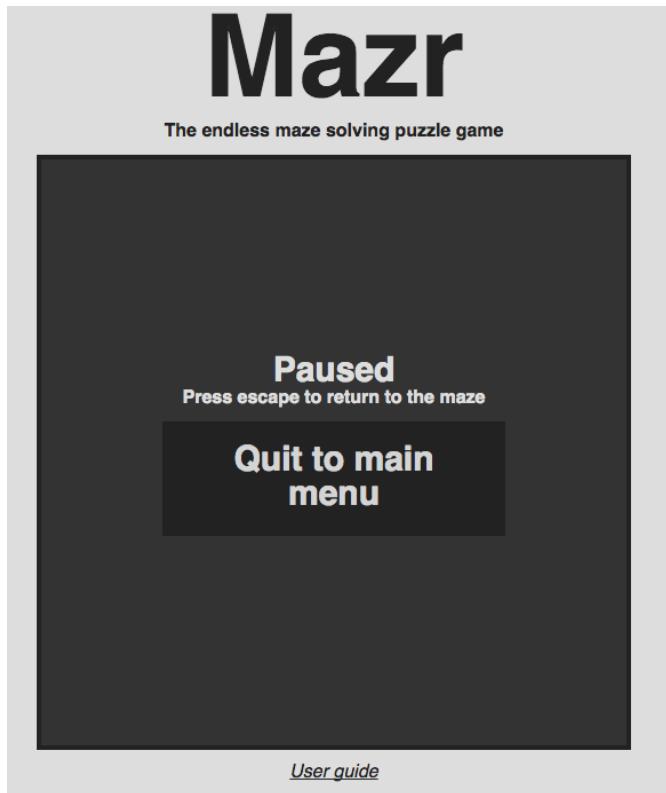
Upon entering the up left arrow key, then the up arrow key, the player's coloured tile moved to the position shown in the leftmost image below. It left an observably correct colour trail. This showed that input to the system through the use of the 'keydown' JavaScript event listener worked correctly at this time. The rightmost image shows further player movement towards the orange exit tile.



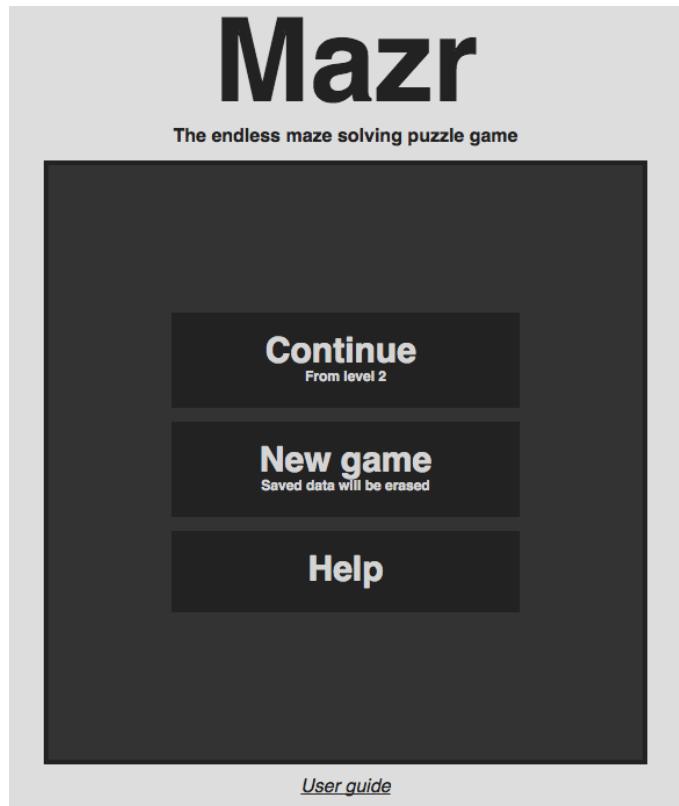
Pressing the next level button revealed level two. Below is an image of the 6x6 maze along with a player trail caused by me using the arrow keys to move the player around the maze. Note I left the game unattended for a small period of time and the colour of the player changed. A more rigorous inspection of the colour change will be observed in Testing.



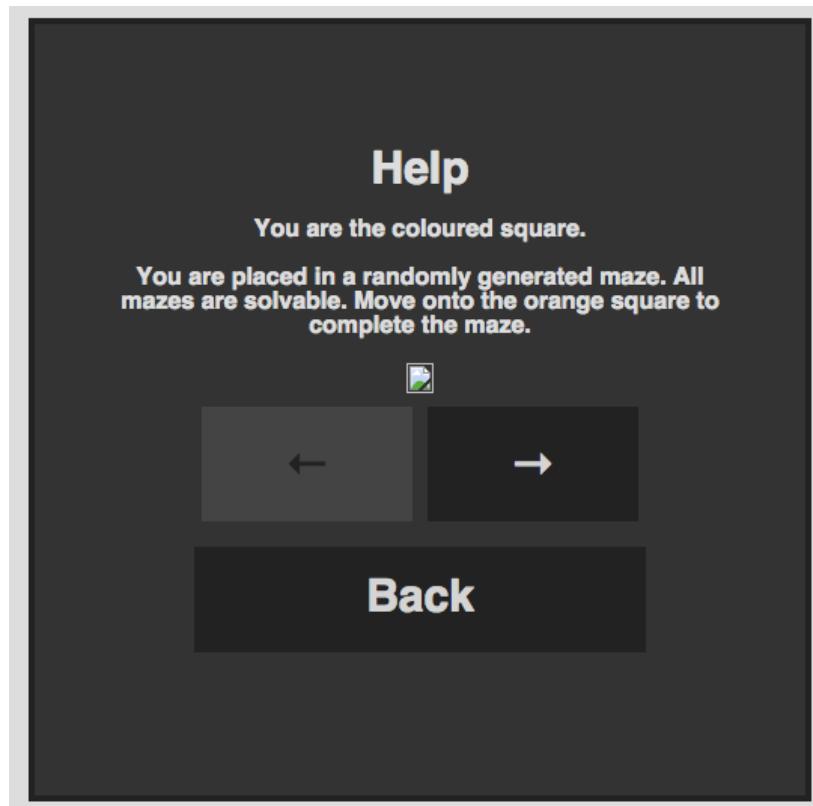
By pressing the escape key, I entered the pause menu. This is shown in the image below.



I pressed the 'Quit to main menu' button. This returned me to the main menu so I could observe the functionality of the continue button. This is shown in the image below. Note it says continue from level 2 – this is correct because having finished level 1 the game continues onto level 2 when the button is pressed.



It was now time to check the help menus. Pressing the help button revealed the following menu page. It is immediately apparent that there is a missing image link on the help page. This is because the help images, which will indeed be screenshots from the game, haven't yet been created and inserted in the correct place.



**Inserting the help images**

Therefore, the three help images must be created. To do this, screenshots were taken of the player at various stages of the game in order to accompany the captions on each of the menus. Below are images of the three screenshots that were taken.

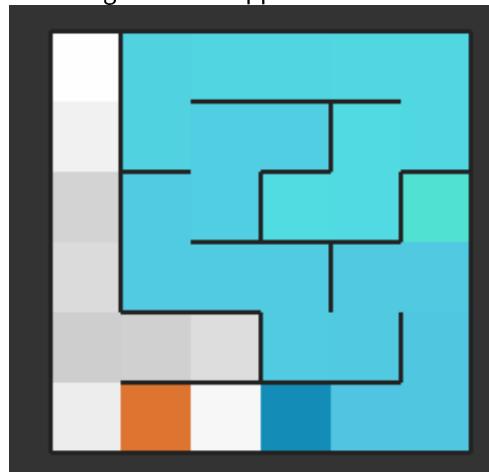
This image that will appear on the first help menu:



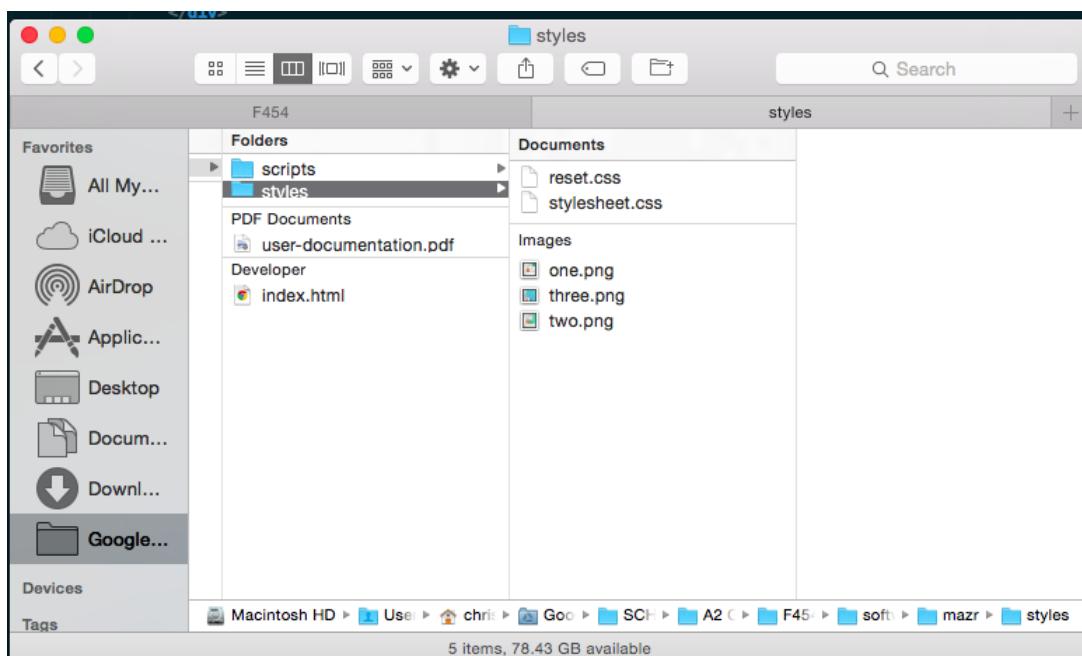
This image that will appear on the second help menu:



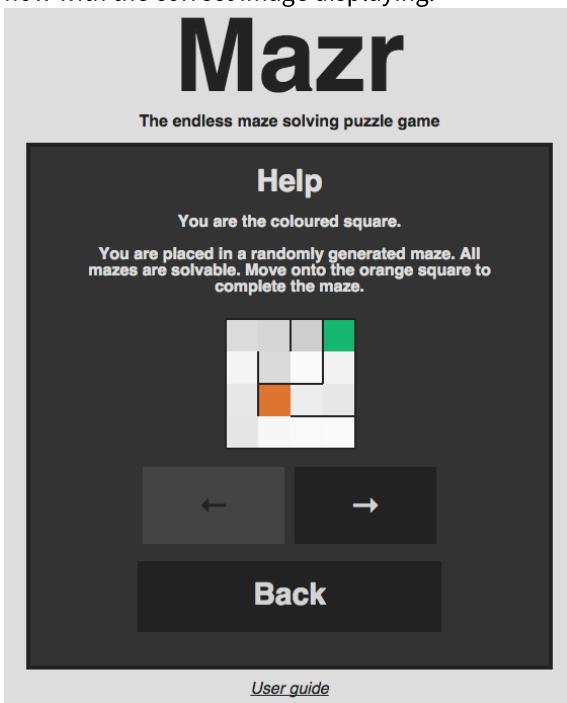
This image that will appear on the third help menu:



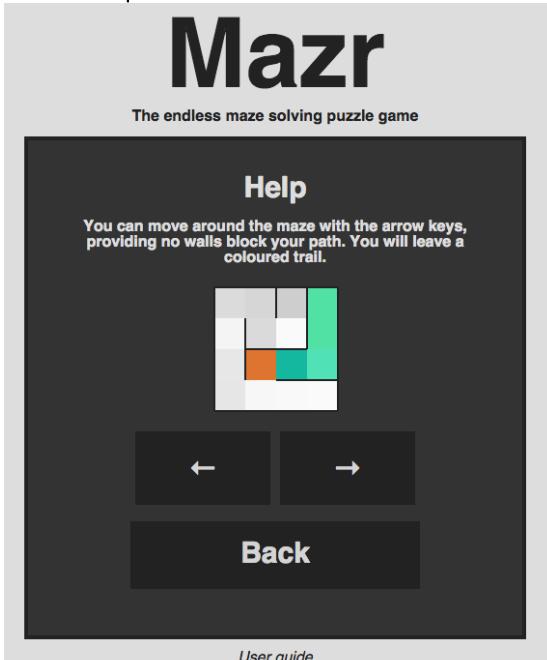
There were given the filenames one.png, two.png and three.png respectively. These were then copied into the styles folder of the game so they could be linked to correctly by the HTML. The image below shows a screenshot of them in their correct location.



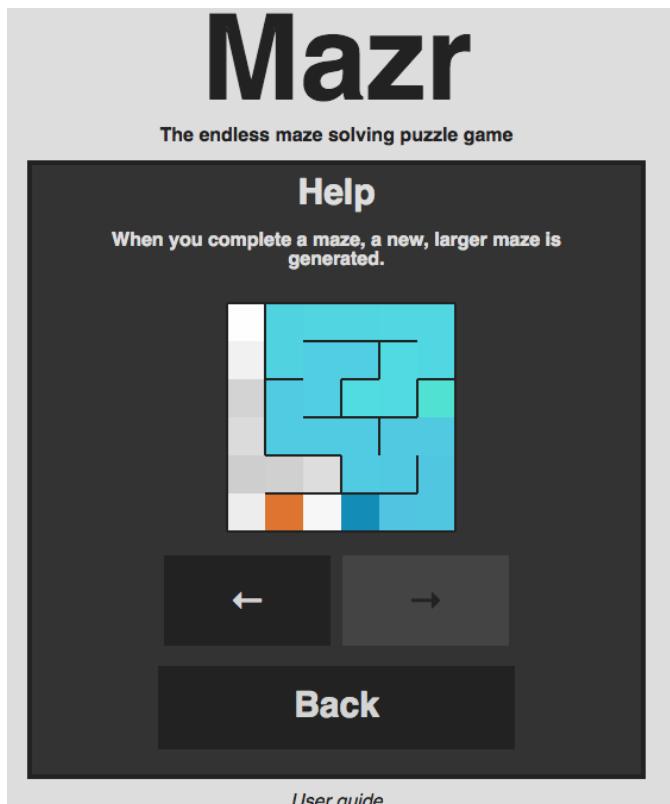
With this change made, the help menus were revisited. The first help menu is shown again below, now with the correct image displaying.



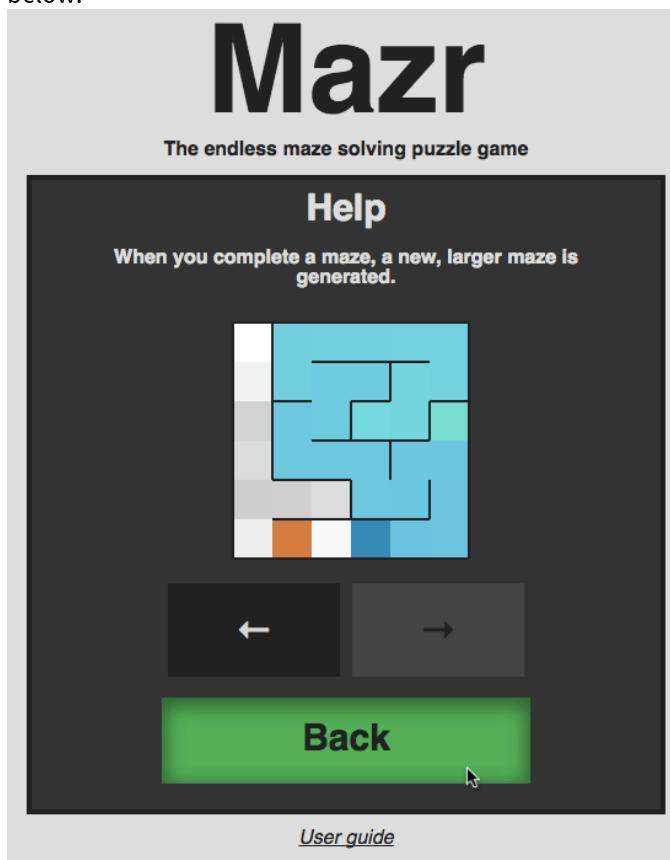
Note the left arrow button is disabled but pressing the right arrow button navigates the user to the second help menu.

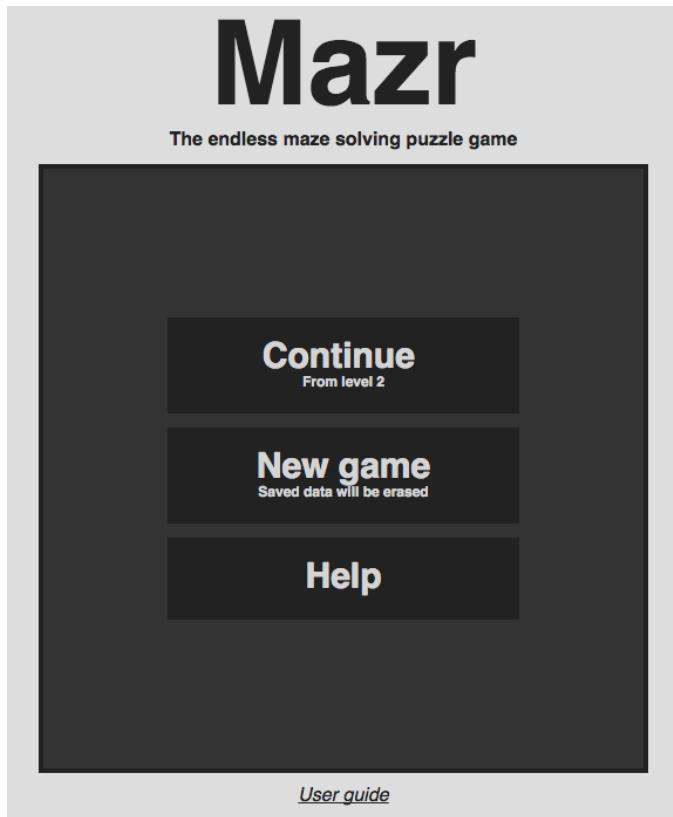


On the second help menu both arrows are enabled and therefore can be used to navigate back to the first help menu in the case of the left arrow button or to the third help menu in the case of the right arrow button. Pressing the right arrow button indeed navigates to the third help menu, which is shown in the image below.



The third help menu has a left arrow button that is enabled so as to return the user to the second help menu. The right arrow button is disabled because there are no further help menus to navigate to. The back button on all three menus navigates back to the main menu as shown in the images below.





This concludes the alpha testing stage of the engine.js file.

## Complete program listings

[Camera.js](#)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <title>Mazr</title>

    <!-- Link to stylesheets -->
    <link rel="stylesheet" href="styles/reset.css" />
    <link rel="stylesheet" href="styles/stylesheet.css" />

    <!-- Include library -->
    <script src="scripts/lib/jquery-2.1.3.min.js" type="text/javascript"></script>

    <!--[if lt IE 9]>
        <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
</head>
<body>
    <hgroup>
        <h1>Mazr</h1>
        <p>The endless maze solving puzzle game</p>
    </hgroup>

    <noscript>
        You need to enable JavaScript in your browser in order to play the game.
    </noscript>

    <section>
        <div class="canvas-wrapper">
            <canvas id="canvas" class="page">
                Unfortunately, your browser won't run Mazr. Please see our list
                of compatible browsers.
            </canvas>

            <div class="main-menu page">
                <div class="buttons">
                    <button tabindex="1" class="btn-continue">Continue</button>
                    <button tabindex="2" class="btn-new-game">New game<br>
                        <p class="under-text">Saved data will be erased</p>
                    </button>
                    <button tabindex="3" class="btn-help">Help</button>
                </div>
            </div>

            <div class="help-1 page">
                <div class="buttons">
                    <h2>Help</h2><br>

```

<p>You are the coloured square. </p><br>

<p> You are placed in a randomly generated maze.  
All mazes are solvable. Move onto the orange  
square to complete the maze.

</p><br>



<div class="arrows">  
    <button disabled class="btn-arrow left">=←</button>  
    <button class="btn-arrow right">=薔</button>  
    </div>

<button class="btn-back">Back</button>

</div>

<div class="help-2 page">  
    <div class="buttons">  
        <h2>Help</h2><br>  
  
        <p> You can move around the maze with the arrow  
keys, providing no walls block your path.  
You will leave a coloured trail.  
</p><br>  
  
          
        <div class="arrows">  
            <button class="btn-arrow left">=←</button>  
            <button class="btn-arrow right">=薔</button>  
            </div>  
        <button class="btn-back">Back</button>  
    </div>  
</div>

<div class="help-3 page">  
    <div class="buttons">  
        <h2>Help</h2><br>  
  
        <p> When you complete a maze, a new, larger maze  
is generated.  
</p><br>  
  
          
        <div class="arrows">

```

        <button class="btn-arrow
left">&#x2190;</button>
        <button disabled class="btn-arrow
right">&#8594;</button>
    </div>

        <button class="btn-back">Back</button>
    </div>
</div>

<div class="pause-menu page">
    <div class="buttons">
        <h2>Paused</h2>
        <p>Press escape to return to the maze</p>
        <button class="btn-quit">Quit to main menu</button>
    </div>
</div>

<div class="end-of-level-menu page">
    <div class="buttons">
        <h2>Congratulations!</h2>
        <p>You completed the maze in <span
id="time"></span></p>
        <p>You took <span id="steps"></span> steps to
complete the maze.</p>
        <p>Your progress has been saved.</p>
        <button class="btn-next-level">Next level</button>
        <button class="btn-quit">Quit to main menu</button>
    </div>
</div>

    </div>
</section>

<footer>
    <a href="user-documentation.pdf">User guide</a>
</footer>

<!-- Include all the necessary scripts -->
<script src="scripts/camera.js" type="text/javascript"></script>
<script src="scripts/sprite.js" type="text/javascript"></script>
<script src="scripts/SpriteBatch.js" type="text/javascript"></script>
<script src="scripts/timer.js" type="text/javascript"></script>
<script src="scripts/player.js" type="text/javascript"></script>
<script src="scripts/ExitTile.js" type="text/javascript"></script>
<script src="scripts/maze.js" type="text/javascript"></script>
<script src="scripts/engine.js" type="text/javascript"></script>
</body>
</html>

```

**Stylesheet.css**

```

html {
    box-sizing: border-box;
}
*, *:before, *:after {
    box-sizing: inherit;
}

noscript {
    font-size: 28px;
    font-weight: normal;
    font-style: italic;
    color: red;
    width: 512px;
    display: block;
    margin: auto;
}

body {
    background: #ddd;
    color: #222;

    font-family: Helvetica;
    font-weight: bold;
    text-align: center;

    margin-left: auto;
    margin-right: auto;
}

hgroup h1 {
    font-size: 100px;
}
hgroup p {
    font-size: 16px;
}

hgroup, footer {
    padding: 12px 0px;
}

h2 {
    font-size: 30px;
}

p {
    padding: 0px 50px;
}

a {
    color: #222222;
    font-style: italic;
    font-weight: lighter;
}

.under-text {
    font-size: 12px;
}

canvas {
    border: 4px solid #222;
}

.page {
    background: #333;
    display: inline-flex;
    border: 4px solid #222;

    color: #ddd;
    width: 520px;
    height: 520px;
    margin: auto;
}

canvas, .page {
    padding: 0;
    margin-bottom: 0;
}

.buttons {
    margin: auto;
}

.btn-arrow {
    display: inline;
    width: 150px;
    border: 5px solid #333333;
    margin: 0;
}

button {
    display: block;
    margin: 12px auto;
    padding: 8px 12px;

    width: 300px;
    background: #222;
    border: 0px;
    padding: 20px;

    /* Text */
    font-size: 30px;
    font-family: Helvetica;
    font-weight: bold;
    color: #d4d4d4;
}

button:active {
    outline: none;

    -webkit-box-shadow: inset 0px 4px 20px 4px
        rgba(0,0,0, 0.4);
    -moz-box-shadow: inset 0px 4px 20px 4px
        rgba(0,0,0, 0.4);
    box-shadow: inset 0px 4px 20px 4px
        rgba(0,0,0, 0.4);
}

button:focus {
    outline: none;

    color: #222;
    background: hsl(145, 80%, 40%);
}

button:disabled {
    outline: none;

    background: #444;
    color: #222;
}

```

**Camera.js**

```

/*
camera.js
Camera object
-----
Written on the 29/10/2014 by Chris Jones.

This file represents the Camera data structure represented in the Data
Structures section of the design specification.

The function Camera represents a function from which a Camera object can be
instantiated. This will represent the Camera data structure.

This class is used by the Maze object to keep track of the current position
of the player's view at any given time. It also allows the focus of the maze
to be adjusted and not fixed at a given position.

The camera has 4 attributes:
    - x: The pixel coordinate of the camera's position in the x axis.
    - y: The pixel coordinate of the camera's position in the y axis.
    - width: The width of the camera's viewport in pixels
    - height: The height of the camera's viewport in pixels.

*/
function Camera(x, y, width, height) {

    // Check if x supplied is of primitive type Number
    if (typeof x === "number") {
        // Assign value of parameter x to attribute x
        this.x = x;
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (x === undefined || x === null) {
        // Assign x attribute default value of 0
        this.x = 0;
    }
    else {
        // Throw an error if above conditions are not met
        // Note errors will be caught in the main game loop.
        throw "Argument error. x coordinate must a number.";
    }

    // Check if y supplied is of primitive type Number
    if (typeof y === "number") {
        // Assign value of parameter y to attribute y
        this.y = y;
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (y === undefined || y === null) {
        // Assign x attribute default value of 0
        this.y = 0;
    }
    else {
        // Throw an error if above conditions are not met.
        // Note errors will be caught in the main game loop.
        throw "Argument error. y coordinate must a number."
    }
}

```

```

        }

        // Check if the width supplied is of primitive type Number
        if (typeof width === "number") {
            // Assign value of parameter width to attribute width
            this.width = width;
        }
        // Check if no parameter was passed (undefined) or a null parameter was passed
        else if (width === undefined || width === null) {
            // Assign width attribute default value of 512
            this.width = 512;
        }
        else {
            // Throw an error if above conditions are not met.
            // Note errors will be caught in the main game loop.
            throw "Argument error. Width must a number."
        }

        // Check if the height supplied is of primitive type Number
        if (typeof height === "number") {
            // Assign value of parameter height to attribute he
            this.height = height;
        }
        // Check if no parameter was passed (undefined) or a null parameter was passed
        else if (height === undefined || height === null) {
            // Assign height attribute default value of 512
            this.height = 512;
        }
        else {
            // Throw an error if above conditions are not met.
            // Note errors will be caught in the main game loop.
            throw "Argument error. Height must a number."
        }

    }
}

```

**Sprite.js**

```

/*
sprite.js
Sprite object
-----
Written on the 29/10/2014 by Chris Jones.

```

This file represents the Sprite data structure represented in the Data Structures section of the design specification.

The function `Sprite` represents a function from which the `Sprite` object can be instantiated. This will represent the sprite data structure.

The `sprite` object represents a coloured square with 4 optional walls (borders) that can be drawn to the screen.

This is used by the `Sprite Batch` to maintain an array of objects that have data which can be drawn to the screen. It exposes properties such as coordinates and colour in order to do this.

Sprite has 4 attributes:

- colour: The colour with which the sprite is drawn. This is a string that would be used to define a colour in CSS for example "#000000" or "rgb(255, 0, 255)".
- x: The pixel coordinate of the sprite's position in the x axis.
- y: The pixel coordinate of the sprite's position in the y axis.
- walls: This is an array that contains 4 boolean values (true or false). This represents whether there is a wall on each side of the square sprite. The array represents each side in a clockwise order (e.g. [north, east, south, west]) For example a walls attribute with value [true, false, true, false] has a north and south wall but no east or west wall.

```
/*
function Sprite(colour, x, y, walls) {

    // Check if x supplied is of primitive type Number
    if (typeof x === "number") {
        // Assign value of parameter x to attribute x
        this.x = x;
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (x === undefined || x === null) {
        // Assign x attribute default value of 0
        this.x = 0;
    }
    else {
        // Throw an error if above conditions are not met
        // Note errors will be caught in the main game loop.
        throw "Argument error. x coordinate must a number.";
    }

    // Check if y supplied is of primitive type Number
    if (typeof y === "number") {
        // Assign value of parameter y to attribute y
        this.y = y;
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (y === undefined || y === null) {
        // Assign x attribute default value of 0
        this.y = 0;
    }
    else {
        // Throw an error if above conditions are not met.
        // Note errors will be caught in the main game loop.
        throw "Argument error. y coordinate must a number."
    }

    // Set the colour of the sprite
    if (typeof colour === "string") {
        // Assign value of parameter colour to attribute colour
    }
}
```

```
        this.colour = colour;
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (colour === undefined || colour === null) {
        // Assign colour attribute default value of "#000000" (black)
        this.colour = "#000000";
    }
    else {
        // Throw an error if above conditions are not met.
        // Note errors will be caught in the main game loop.
        throw "Argument error. colour must a string."
    }

    // Set the value of the attribute walls
    if (walls instanceof Array && walls.length == 4) {
        // Declare local variable correctType as true
        // Assume at this point the elements of walls are all boolean data-types
        var correctType = true;

        // Iterate through each element of walls with index i
        for (var i = 0; i < walls.length; i++) {
            // Check whether the type is NOT boolean
            if (typeof walls[i] != "boolean") {
                // Set correctType to false if not a boolean
                correctType = false;
                // Break from the loop if a non-boolean type is detected
                break;
            }
        }

        // If correctType is true (when all 4 elements are boolean)
        if (correctType)
            // Assign parameter walls to attribute walls
            this.walls = walls;
        else
            // Throw an error at this point if not all data types are boolean
            // Note errors will be caught in the main game loop.
            throw "Argument error. walls must be an array with 4 boolean elements"
    }
    // Check if no parameter was passed (undefined) or a null parameter was passed
    else if (walls === undefined || walls === null) {
        // Assign walls attribute default value of [false, false, false, false]
        this.walls = [false, false, false, false];
    }
    else {
        // Throw an error if above conditions are not met.
        // Note errors will be caught in the main game loop.
        throw "Argument error. walls must be an array with 4 boolean elements"
    }
}
```

**SpriteBatch.js**

```
/*
SpriteBatch.js
SpriteBatch object
-----
Written on the 30/10/2014 by Chris Jones
```

This file represents the SpriteBatch data structure represented in the Data Structures section of the design specification.

The function SpriteBatch represents a function from which the SpriteBatch object can be instantiated. This will represent the spriteBatch data structure.

SpriteBatch controls the drawing of an array of sprites to a canvas through a canvas' drawing context. It maintains an array of sprites that are drawn one by one to the canvas in their correct position by drawing coloured rectangles. Once all of the rectangles are drawn then paths are created for each of the sprite's walls. This is encapsulated in a draw function

A SpriteBatch object is maintained by a Maze object in order to draw all the sprites generated by a maze. It exposes its drawing functions so it draws to the canvas approximately 30 times a second.

SpriteBatch has 3 attributes:

- batch: An array, declared empty upon instantiation, that contains the sprites that are to be drawn to the canvas. Sprites are added in order relative to their z-index on the canvas. For example, adding two sprites with exactly the same positions will cause the last sprite added to be drawn on top of the first sprite added.
- canvasContext: A reference to the 2D drawing context of the canvas onto which the sprites will be drawn.
- camera: Reference to the camera object with which the game is being viewed.

SpriteBatch has one method:

- draw: This draws all of the sprites in batch to canvas serially (one after another). All of the sprite's backgrounds are first drawn and then the walls are all drawn together on top. This ensures no sprites are drawn on top of the walls.

```
/*
function SpriteBatch(canvasContext, camera) {
    // Contains a batch of sprites to draw
    // Initialise as an empty array
    // As this is an attribute of the SpriteBatch object
    // it can be populated later
    this.batch = [];

    // Reference to canvasContext on which the batch is drawn
    // Note there is no defaulting code path for the canvasContext
    // This is because it would be meaningless for a spritebatch
    // to have a canvas drawing context defined itself
```

```

if (canvasContext === undefined || canvasContext === null) {
    throw "Argument error. Improper canvas context specified.";
} else {
    // Assign canvasContext parameter to canvasContext attribute
    this.canvasContext = canvasContext;
}

// Reference to camera to which the sprites will be transformed
// Note there is no defaulting code path for the camera
// This is because it would be meaningless for a spritebatch
// to draw to its own camera
if (camera === undefined || camera === null) {
    throw "Argument error. Improper camera specified.";
} else {
    // Assign camera parameter to camera attribute
    this.camera = camera;
}

// draw function acts as method that handles the drawing of sprites in batch
this.draw = function() {
    // Iterate through each sprite in the batch
    for (var i in this.batch) {
        // Check that the i'th sprite is located
        // within the bounds of the camera
        // This means computer resources aren't wasted rendering
        // a sprite that isn't going to be seen

        // Subtracting the camera's coordinate's from those of
        // the sprite effectively moves the sprites with respect
        // to a stationary camera. This has the effect of the camera
        // moving and the sprites remaining stationary
        if ((this.batch[i].x - this.camera.x < this.camera.width) &&
            (this.batch[i].y - this.camera.y < this.camera.height)) {

            // Set the fillStyle of the canvas equal
            // to the colour of the sprite
            this.canvasContext.fillStyle = this.batch[i].colour;

            // Draw a rectangle onto the canvas
            // At the x and y coods of the sprite minus the camera's coods
            // The size of the rectangle should equal
            // the TILE_SIZE constant
            this.canvasContext.fillRect(
                this.batch[i].x - this.camera.x,
                this.batch[i].y - this.camera.y,
                TILE_SIZE,
                TILE_SIZE
            );
        }
    }

    // Set the colour of the walls to the constant WALL_COLOUR
    this.canvasContext.strokeStyle = WALL_COLOUR;
    // Set the width of the walls to constant TILE_SIZE divided by 16
    this.canvasContext.lineWidth = TILE_SIZE / 16;
}

```

```

// Begin drawing the path
this.canvasContext.beginPath();

// Draw the walls
for (var i in this.batch) {
    // Check that the i'th sprite is located
    // within the bounds of the camera
    // This means computer resources aren't wasted rendering
    // a sprite that isn't going to be seen

    // Subtracting the camera's coordinate's from those of
    // the sprite effectively moves the sprites with respect
    // to a stationary camera. This has the effect of the camera
    // moving and the sprites remaining stationary
    if ((this.batch[i].x - this.camera.x < this.camera.width) &&
        (this.batch[i].y - this.camera.y < this.camera.height)) {

        // Check if there is a top / north wall and draw if there is
        if (this.batch[i].walls[0]) {
            // Move the starting point to the top left-hand corner of
            // ith sprite
            this.canvasContext.moveTo(
                this.batch[i].x - this.camera.x,
                this.batch[i].y - this.camera.y
            );

            // Draw a line to the top right-hand corner of ith sprite
            this.canvasContext.lineTo(
                this.batch[i].x - this.camera.x + TILE_SIZE,
                this.batch[i].y - this.camera.y
            );
        }

        // Check if there is a right / east wall and draw if there is
        if (this.batch[i].walls[1]) {
            // Move to the top-right hand corner of ith sprite
            this.canvasContext.moveTo(
                this.batch[i].x - this.camera.x + TILE_SIZE,
                this.batch[i].y - this.camera.y
            );

            // Draw a line to the bottom-right hand corner of ith
            // sprite
            this.canvasContext.lineTo(
                this.batch[i].x - this.camera.x + TILE_SIZE,
                this.batch[i].y - this.camera.y + TILE_SIZE
            );
        }

        // Check if there is a bottom / south wall and draw if there is
        if (this.batch[i].walls[2]) {
            // Move to the bottom-right hand corner of ith sprite
            this.canvasContext.moveTo(
                this.batch[i].x - this.camera.x + TILE_SIZE,

```

```

        this.batch[i].y - this.camera.y + TILE_SIZE
    );
}

// Draw a line to the bottom-left hand corner of ith
// sprite
this.canvasContext.lineTo(
    this.batch[i].x - this.camera.x,
    this.batch[i].y - this.camera.y + TILE_SIZE
);
}

// Check if there is a left / west wall and draw if there is
if (this.batch[i].walls[3]) {
    // Move to the bottom-left hand corner of ith sprite
    this.canvasContext.moveTo(
        this.batch[i].x - this.camera.x,
        this.batch[i].y - this.camera.y + TILE_SIZE
    );

    // Draw a line to the top-left hand corner of ith sprite
    this.canvasContext.lineTo(
        this.batch[i].x - this.camera.x,
        this.batch[i].y - this.camera.y
    );
}
}

// Draw the walls to canvasContext with .stroke() function
this.canvasContext.stroke();
}
}

```

**timer.js**

```
/*
timer.js
Timer object
-----
```

Written on the 02/01/2014 by Chris Jones

This is a class function that will be used to instantiate Timer objects. These represent the Timer data structures in the nature of the solution section of Design.

Timers are used in a variety of methods and objects in the game to keep track of timing. This is crucial to a variety of areas including camera panning, player animation and the fading of the player's colour.

Timer has 2 local variables:

- lastTime: Stores the current number of ms at the last call of the run method.
- elapsedTime: Stores the amount of time that has elapsed since the last call of the run method.

Timer has 1 method:

- run: This method will only return true when the Number parameter wait\_time has elapsed (wait\_time is in milliseconds). To call something periodically, call it with a constant wait\_time enclosed in an if loop that has a boolean conditional of the return value of this run method.

```
/*
function Timer() {

    // Declare lastTime with the value of the time when timer is initialised
    // Set lastTime equal to the current milliseconds on timer
    var lastTime = new Date().getTime();

    // elapsedTime stores the time taken between calls of the run method
    // Set to zero on declaration
    var elapsedTime = 0;

    // Method returns true every period of 'wait_time'
    // For this to work this method must be polled
    // By repeatedly calling the run method and over given intervals,
    // time changes can be tracked
    this.run = function(wait_time) {
        // Set now to current ms
        var now = new Date().getTime();

        // Add the difference between now and last check of ms to elapsedTime
        elapsedTime += (now - lastTime);
        // Let lastTime equal now
        lastTime = now;

        // If the elapsedTime since last poll of function
        // is greater than the wait
        if (elapsedTime >= wait_time) {
            // Set elapsedTime to zero
            elapsedTime = 0;
            // Return a true value
            return true;
        }
    }
}
```

### player.js

```
/*
player.js
Player object
-----
Written on the 31/10/2014 by Chris Jones
```

This file represents the Player data structure represented in the Data Structures section of the design specification.

The function Player represents a function from which the Player object can be instantiated. This will represent the Player data structure.

A player object encapsulates all of the functionality necessary with introducing a player into the game. It exposes getter and setters to make accessing the position of the player more expressive for later programming and also ensures the player is correctly coloured and animated.

A player object is instantiated by the Maze object and controlled within that object's functionality. This allows it to be added to a SpriteBatch, thus be drawn by that and updated by the Maze.

Player has 4 attributes:

- disabled: This is a boolean value that is used to control whether the player can move or not. This is so that the player's movement can be disabled when the camera is panning from the exit tile or currently on a menu.
- sprite: This is the object that is responsible for the visuals and drawing of the player to the canvas. This attribute will be referenced to within a SpriteBatch.
- newPositionX: This is a Number that represents an x coordinate. This is used to track the exact pixel coordinates of a player in the maze instead of relative to other grid squares. Doing this allows smooth animations to be implemented.
- newPositionY: This is a Number that represents a y coordinate. This is used to track the exact pixel coordinates of a player in the maze instead of relative to other grid squares. Doing this allows smooth animations to be implemented.

Player has 2 pairs of getter / setter methods:

- x: This is a getter / setter pair that reference the x coordinate of the sprite. This is effectively assigning the reference player.x to player.sprite.x. This makes it easier to get and set values of the player's position without accessing the sprite.
- y: This is a getter / setter pair that reference the y coordinate of the sprite. This is effectively assigning the reference player.y to player.sprite.y. This makes it easier to get and set values of the player's position without accessing the sprite.

Player has 3 methods:

- animatePosition: This method updates the position of the player on the screen. This is so that it moves smoothly from tile to tile in maze.
- fadeColour: This returns a HSL CSS string that varies with respect to time. This can be used to be assigned to the colour values of objects in the maze - in this case the player's sprite.

- trailColour: Returns a HSL CSS string that is the colour that tiles on which the player is leaving a trail should be coloured.

Player has 8 local variables:

- hueOne: This is the initial hue of the fading colour.
- hueTwo: This is the hue to which the fading colour fades.
- currentHue: The current hue of the player at any given time.
- dHue: A change in hue. This is a unit of change in hue with one step in time.
- currentSaturation: The current saturation of the fading colour.
- currentLightness: The current lightness of the fading colour.
- fadeDirection: The fading colour will fade from hueOne to hueTwo and then back again in a periodic cycle. This is a boolean that stores in which direction the fading is occurring.
- fadingTimer: This is a Timer object that keeps track of the timing with respect to which the colour fades.

```
*/
function Player(sprite) {
    // Define getter and setter for y position property
    // These make it easier to reference the position of player
    this.__defineGetter__("x", function() {
        // Return the value of the sprite's x property
        return this.sprite.x;
    });
    this.__defineSetter__("x", function(x) {
        // Set the value of the sprite's x property to the parameter x
        this.sprite.x = x;
    });

    // Define getter and setter for y position property
    // These make it easier to reference the position of player
    this.__defineGetter__("y", function() {
        // Return the value of the sprite's y property
        return this.sprite.y;
    });
    this.__defineSetter__("y", function(y) {
        // Set the value of the sprite's y property to the parameter y
        this.sprite.y = y;
    });

    // Set the disabled property of the player to false by default
    // When disabled is true movement of player is restricted
    // This occurs in the maze.js position updating methods
    this.disabled = false;

    // Check if sprite parameter is of type Sprite
}
```

```

if (sprite instanceof Sprite) {
    // Assign value to parameter sprite to property sprite
    this.sprite = sprite;
    // Otherwise throw an error
} else {
    throw "Argument error. sprite must be a Sprite object."
}

// We know by validation in the sprite class its properties must be correct
// Therefore we can assign new values here
this.newPositionX = this.x;
this.newPositionY = this.y;

// This method updates the position of the player on the screen
// This is so that it moves smoothly from tile to tile in maze
this.animatePosition = function() {
    // Update the x position of the sprite in small increments
    // This ensures the player moves smoothly
    if (this.newPositionX > this.x) {
        // Increase x coordinate by quarter of the size of game tiles
        this.x += TILE_SIZE/4;
    } else if (this.newPositionX < this.x) {
        // Decrease x coordinate by quarter of the size of tiles
        this.x -= TILE_SIZE/4;
    }
}

// Update the y position of the sprite in small increments
// This ensures the player moves smoothly
if (this.newPositionY > this.y) {
    // Increase y coordinate by quarter of the size of game tiles
    this.y += TILE_SIZE/4;
} else if (this.newPositionY < this.y) {
    // Decrease y coordinate by quarter of the size of tiles
    this.y -= TILE_SIZE/4;
}
}

// This is the initial hue of the fading colour.
var hueOne = 145;
// This is the hue to which the fading colour fades.
var hueTwo = 300;
// The current hue of the player at any given time.
var currentHue = hueOne;
// A change in hue. This is a unit of change in hue with one step in time.
var dHue = (hueTwo - hueOne)/2000;
// The current saturation of the fading colour.
var currentSaturation = 80;
// The current lightness of the fading colour.
var currentLightness = 40;
// Stores the direction of fade of colour
var fadeDirection=true;
// Keeps track of the timing with respect to which the colour fades.
var fadingTimer = new Timer();

// Function handles colour fading of player's colour

```

```

this.fadeColour = function() {
    // Make change every 0.001 seconds
    if (fadingTimer.run(1)) {
        // Choose direction of change
        // If the currentHue is greater than the second hue move backwards
        if (currentHue >= hueTwo) { fadeDirection = false; }
        // Otherwise if the currentHue is less than the
        // first hue then move forwards
        else if (currentHue <= hueOne) { fadeDirection = true; }

        // If fadeDirection is forward
        if (fadeDirection) {
            // Increment the currentHue by the change in hue
            currentHue += dHue;
        } else {
            // Decrement the currentHue by the change in hue
            currentHue -= dHue;
        }
    }

    // Update the sprite with respect to the change
    return "hsl(" + parseInt(currentHue) + "," +
        currentSaturation + "%," +
        currentLightness + "%)";
}
}

this.trailColour = function() {
    return "hsl(" + currentHue + "," + (currentSaturation - 10) + "%," + (currentLightness + 20) + "%)";
}
}

```

[ExitTile.js](#)

```

/*
ExitTile.js
ExitTile object
-----
Written on the 01/01/2014 by Chris Jones

```

This file represents the ExitTile data structure represented in the Data Structures section of the design specification.

The function ExitTile represents a function from which the ExitTile object can be instantiated. This will represent the ExitTile data structure.

An ExitTile represents the end of a maze - the tile to which the player must move in order to complete any given maze. Therefore, it must be drawn to the maze as would any sprite.

An ExitTile object is instantiated by the Maze object and controlled within the body of that object. This allows for it to be controlled and its position used in conjunction with the Player's in order to determine if the maze has been completed.

ExitTile has 1 attribute:

- sprite: The sprite object that is drawn to the canvas that represents the ExitTile in the maze.

ExitTile has 2 pairs of getter / setter methods:

- x: This is a getter / setter pair that reference the x coordinate of the sprite. This is effectively assigning the reference ExitTile.x to ExitTile.sprite.x. This makes it easier to get and set values of the ExitTile's position without accessing the sprite.
- y: This is a getter / setter pair that reference the y coordinate of the sprite. This is effectively assigning the reference ExitTile.y to ExitTile.sprite.y. This makes it easier to get and set values of the ExitTile's position without accessing the sprite.

\*/

```
function ExitTile(sprite) {
    // Define getter and setter for x position property
    // These make it easier to reference the position of an ExitTile
    this.__defineGetter__("x", function() {
        // Return the value of the sprite's x property
        return this.sprite.x;
    });
    this.__defineSetter__("x", function(x) {
        // Set the value of the sprite's x property to the parameter x
        this.sprite.x = x;
    });

    // Define getter and setter for y position property
    // These make it easier to reference the position of an ExitTile
    this.__defineGetter__("y", function() {
        // Return the value of the sprite's y property
        return this.sprite.y;
    });
    this.__defineSetter__("y", function(y) {
        // Set the value of the sprite's y property to the parameter y
        this.sprite.y = y;
    });

    // Check if sprite parameter is of type Sprite
    if (sprite instanceof Sprite) {
        // Assign value to parameter sprite to property sprite
        this.sprite = sprite;
    } else {
        throw "Argument error. sprite must be a Sprite object."
    }
}
```

/\*

maze.js  
Maze object

-----  
Written on the 14/11/2014 by Chris Jones

[maze.js](#)

This file represents the Maze data structure represented in the Data Structures section of the design specification.

The function Maze represents a function from which the Maze object can be instantiated. This will represent the Maze data structure.

A maze object encapsulates all of the functionality necessary with generating, updating and drawing a maze and all of its constituents, including the player and exit tile.

The Maze is instantiated in the global scope and used to control the maze that is generated. The maze is reinstantiated at a larger size when a previous maze is completed.

Maze has 13 attributes:

- complete: Boolean value that stores whether the maze is completed or not.
- maxX: Stores the maximum number of tiles wide (in the horizontal direction) the maze will be.
- maxY: Stores the maximum number of tiles high (in the vertical direction) the maze will be.
- camera: A reference to the camera through which the maze is viewed. Used to control view of the player / panning.
- tiles: An array that stores the tiles as Sprite objects in the maze.
- player: Reference to the Player object that will be controlled in the maze.
- exitTile: Reference to the ExitTile object that will be set as the player's goal.
- spriteBatch: Reference to a SpriteBatch object that will be instantiated and used to draw the maze to the canvas.
- animationTimer: Controls timing of player movement animations.
- movementTimer: Controls timing of the movement of player.
- panningTimer: Controls the panning of the camera.
- completionTimer: Records time taken to complete maze. (in seconds).
- stepsTaken: Stores the number of steps required to complete the maze.

Maze has 2 local variables:

- playerX: A temporary variable that stores a random x coordinate in the maze for the player.
- playerY: A temporary variable that stores a random y coordinate in the maze for the player.
- panning: stores whether the camera is currently independent of player movement.
- fixedMap: Stores whether the level fits within the camera window and therefore disables the camera.

Maze has 5 methods:

```

- initialise: Set all the state to default. Must be called
    when a maze is instiated. This allows maze
    to be re-instantiated with more integrity then
    just a simple instantiation.

- draw: Draws all of the sprites to the canvas.

- update: Updates all of the maze state and data. This needs to
    be called approx. 30 times a second.

- updatePlayerPosition: Called when input is entered into the
    system. Updates the position
of the
player.

- colourTrail: Colours the trail the player leaves when movement
    occurs. Called by updatePlayerPosition when
    movement occurs.

- newMaze: Generates a new maze with a randomised depth first
    search algorithm.

*/
function Maze(width, height, canvasContext, camera) {
    // Boolean stores whether grid is finished
    // Set to false by default
    this.complete = false;

    // Array of tiles that make up grid
    this.tiles = new Array();

    // Set the maze's exit initially to null
    // This is instantiated properly in the init() method
    this.exitTile = null;

    // Various timer objects used to control time-dependent updates
    this.animationTimer = new Timer(); // Controls all animations
    this.movementTimer = new Timer(); // Controls movement of player
    this.panningTimer = new Timer(); // Controls panning of camera
    this.completionTimer = new Timer(); // Records time taken to complete maze

    // Stores the number of seconds taken to complete the maze
}

```

```
this.completedSeconds = 0;

// Stores the number of steps taken in the maze since it has begun
this.stepsTaken = 0;

// Check if the width supplied is of primitive type Number
if (typeof width === "number") {
    // Assign value of parameter width to attribute maxX
    this.maxX = width;
}
// Check if no parameter was passed (undefined) or a null parameter was passed
else if (width === undefined || width === null) {
    // Assign maxX attribute default value of 8
    this.maxX = 8;
}
else {
    // Throw an error if above conditions are not met.
    // Note errors will be caught in the main game loop.
    throw "Argument error. width must a number."
}

// Check if the height supplied is of primitive type Number
if (typeof height === "number") {
    // Assign value of parameter width to attribute maxY
    this.maxY = height;
}
// Check if no parameter was passed (undefined) or a null parameter was passed
else if (height === undefined || height === null) {
    // Assign maxY attribute default value of 8
    this.maxY = 8;
}
else {
    // Throw an error if above conditions are not met.
    // Note errors will be caught in the main game loop.
    throw "Argument error. height must a number."
}

// Reference to camera through which maze is viewed
// Note there is no defaulting code path for the camera
if (camera === undefined || camera === null) {
    throw "Argument error. Improper camera specified.";
}
// Otherwise correctly assign parameter to attribute
} else {
    // Assign camera parameter to camera attribute
    this.camera = camera;
}

// Stores whether the camera is currently panning
// independent of player movement
var panning = true;

// Stores whether the level fits within the camera window and
// therefore disables the camera
var fixedMap = false;
```

```

// The sprite batch that will draw the grid
this.spriteBatch = new SpriteBatch(canvasContext, this.camera);

// Create a random position in the maze for the player
var playerX = Math.floor((Math.random() * this.maxX-1) + 1) * TILE_SIZE;
var playerY = Math.floor((Math.random() * this.maxY-1) + 1) * TILE_SIZE;

// Instantiate a new player attribute
// at the random position (playerX,playerY)
this.player = new Player(
    new Sprite("hsl(145, 80%, 40%)",
        playerX,
        playerY,
        [false,false,false,false]
    )
);

this.initialise = function() {
    // Generate a new random maze
    var maze = this.newMaze();

    // Iterate from zero to max tile size in x direction
    for (var i = 0; i < this.maxX; i++) {
        // Instantiate a nested array of length maxY inside tiles attribute
        this.tiles[i] = new Array(this.maxY);

        // Iterate from zero to max tile size in y direction
        for (var j = 0; j < this.maxY; j++) {
            // Instantiate new sprite at ith,jth index
            this.tiles[i][j] = new Sprite(
                this.randomFloor(),
                i*TILE_SIZE,
                j*TILE_SIZE,
                [maze[i][j][3] == false,
                 maze[i][j][2] == false,
                 maze[i][j][1] == false,
                 maze[i][j][0] == false
                ]
            );
        }
    }

    // Select an acceptable exit zone
    // randomExitX and randomExitY represent random coordinates
    // acceptable being true represents an acceptable set of coordinates
    var randomExitX = 0,
        randomExitY = 0,
        acceptable = false;

    // Repeat until an acceptable value is found
    while (!acceptable) {
        // Generate a random value
        randomExitX = Math.floor((Math.random() * this.maxX-1) + 1);
        randomExitY = Math.floor((Math.random() * this.maxY-1) + 1);
    }
}

```

```

// If the absolute distance between player and exit is third of map, values are acceptable
if (Math.abs(randomExitX - (this.player.x/TILE_SIZE)) > this maxX/3 &&
    Math.abs(randomExitY - (this.player.y/TILE_SIZE)) > this maxY/3) acceptable = true;
}

// Instantiate the exitTile attribute at the coordinates that were generated above
this.exitTile = new ExitTile(
    new Sprite(
        EXIT_COLOUR,
        randomExitX*TILE_SIZE,
        randomExitY*TILE_SIZE,
        [false,false,false,false]
    )
);

// Push all of the cell sprites to the batch
// This is so they can be drawn to canvas
for (var x = 0; x < this.maxX; x++) {
    for (var y = 0; y < this.maxY; y++) {
        this.spriteBatch.batch.push(this.tiles[x][y]);
    }
}

// Push the player and exit tile last
// They will therefore be drawn on top of tiles
this.spriteBatch.batch.push(this.exitTile.sprite);
this.spriteBatch.batch.push(this.player.sprite);

// Determine if the grid wholly fits in the viewport
if (this.maxX * TILE_SIZE <= this.camera.width &&
    this.maxY * TILE_SIZE <= this.camera.height) {
    // If it does, declare the map fixed and centre the grid on the viewport
    panning = false; fixedMap = true;
}

// Centre the camera on the maze
this.camera.x = (this.maxX*TILE_SIZE/2) - (this.camera.width/2);
this.camera.y = (this.maxY*TILE_SIZE/2) -(this.camera.height/2);

} else {
    // Start the camera looking at the exit zone
    // (Camera will then pan to player in update loop)
    this.camera.x = (randomExitX*TILE_SIZE * 2 + TILE_SIZE)/2 - (this.camera.width/2);
    this.camera.y = (randomExitY*TILE_SIZE * 2 + TILE_SIZE)/2 - (this.camera.height/2);
}
}

// Method draws all of the sprites to the canvas
this.draw = function() {
    // Call draw on spriteBatch attribute
    // Draws all the sprites in the batch
    this.spriteBatch.draw();
}

```

```

        }

    // Method updates all of the maze state and data
    this.update = function() {
        // Wait every 1 second (1000ms) for completionTimer to elapse
        if (this.completionTimer.run(1000)) {
            // Increment the second counter by 1 every second
            this.completedSeconds+=1;
        }

        // Update the player's colour
        // This colour changes over time so must be assigned on every update cycle
        this.player.sprite.colour = this.player.fadeColour();

        // If the camera is not currently panning from exit to player
        if (!panning) {
            // Execute every 0.01 seconds (10ms)
            if(this.animationTimer.run(10)) {

                // Animate sprite's position movement
                this.player.animatePosition();
            }

            // Clamp the centre of the camera viewport to the player
            // If the maze isn't small enough to be fixed to the centre
            if (!fixedMap) {
                // Set x and y coordinates of camera to centre of player
                this.camera.x = (2*this.player.x + TILE_SIZE)/2 - (WIDTH/2);
                this.camera.y = (2*this.player.y + TILE_SIZE)/2 - (HEIGHT/2);
            }
            // Else execute this if camera is panning
            } else {
                // Execute every 0.01 seconds (10ms)
                if (this.panningTimer.run(10)) {
                    // Update the x pos of the camera to pan towards player
                    // If the camera has a lower x coordinate than the player
                    // Move the camera right
                    if (this.camera.x < (this.player.x *2 + TILE_SIZE)/2 - (WIDTH/2)) {
                        // Move eighth of width of tile (4px) towards the player in
                        positive x
                        this.camera.x += TILE_SIZE/8;
                    // If the camera has a greater x coordinate than the player
                    // Move the camera left
                    } else if (this.camera.x > (this.player.x *2 + TILE_SIZE)/2 -
                    (WIDTH/2)) {
                        this.camera.x -= TILE_SIZE/8;
                    }

                    // Update the y pos of the camera to pan towards player
                    // If the camera has a lower y coordinate than the player
                    // Move the camera down
                    if (this.camera.y < (this.player.y + this.player.y + TILE_SIZE)/2 -
                    (HEIGHT/2)) {
                        this.camera.y+= TILE_SIZE/8;
                    // If the camera has a greater y coordinate than the player
                    // Move the camera up
                }
            }
        }
    }
}

```

```

        } else if (this.camera.y > (this.player.y + this.player.y +
TILE_SIZE)/2 - (HEIGHT/2)) {
            this.camera.y-= TILE_SIZE/8;
        }

        // Check when the camera is centered on the sprite then cease to
pan
        // Compare position of camera to that of player
        if (this.camera.y == (this.player.y + this.player.y + TILE_SIZE)/2 -
(HEIGHT/2) &&
TILE_SIZE)/2 - (WIDTH/2) ) {
            this.camera.x == (this.player.x + this.player.x +
// Cease to pan when positions are the same
// by setting panning to false
panning = false;
        }
    }

    // Check if the game has been completed
    // Compare position of player with the exit tile
    if (this.player.x == this.exitTile.x &&
        this.player.y == this.exitTile.y) {
        // If they are equal treat the maze as completed
        // Set complete to true
        this.complete = true;
    }
}

// Methods generates a random shade of grey
this.randomFloor = function() {
    // Generate random numerical rgb value
    var g = 255 - Math.floor((Math.random() * 50) + 1);
    // Concatenate into a colour string and return
    return "rgb(" + g + "," + g + "," + g + ")";
}

// Updates the position of the player within the maze
this.updatePlayerPosition = function(direction) {
    if (direction == "up") {
        // Check the player isn't moving by comparing new position to
        // current position and that player is not disabled
        if ((this.player.newPositionY == this.player.y &&
            this.player.newPositionX == this.player.x) &&
            !this.player.disabled ) {

            // Colour in the tile player is on
            this.colourTrail();

            // Check there are no walls blocking the path of the player
            if
(this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[0] == false &&
                this.tiles[this.player.x/TILE_SIZE][(this.player.y-
TILE_SIZE)/TILE_SIZE].walls[2] == false) {

```

```

        // Update the player's new y position
        // Subtract TILE_SIZE from y property of player
        this.player.newPositionY = this.player.y - TILE_SIZE;

        // Increment the number of steps taken
        this.stepsTaken += 1;

    }

}

} else if (direction == "down") {
    // Check the player isn't moving and isn't at bottom of map or disabled
    if ( (this.player.newPositionY == this.player.y &&
          this.player.newPositionX == this.player.x) &&
         !this.player.disabled ) {

        // Colour in the square the player is on
        this.colourTrail();

        // Check there are no walls blocking the path of the player
        if
            (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[2] == false &&
             this.tiles[this.player.x/TILE_SIZE][(this.player.y+TILE_SIZE)/TILE_SIZE].walls[0] == false) {

                // Update the player's new y position
                this.player.newPositionY = this.player.y + TILE_SIZE;
                this.stepsTaken += 1;
            }

        }
    } else if (direction == "left") {
        // Check the player isn't moving and isn't at leftmost y tiles of map or
        disabled
        if ( (this.player.newPositionY == this.player.y &&
              this.player.newPositionX == this.player.x) &&
             !this.player.disabled ) {

            // Colour the square the player is on
            this.colourTrail();

            // Check there are no walls blocking the path of the player
            if
                (this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[3] == false &&
                 this.tiles[(this.player.x-
TILE_SIZE)/TILE_SIZE][this.player.y/TILE_SIZE].walls[1] == false) {

                    // Update the player's new x position
                    this.player.newPositionX = this.player.x - TILE_SIZE;
                    this.stepsTaken += 1;
                }

            }
        } else if (direction == "right") {

```

```

        // Check the player isn't moving and isn't at rightmost y tiles of map or
disabled
if ( (this.player.newPositionY == this.player.y &&
      this.player.newPositionX == this.player.x) &&
      !this.player.disabled ) {

    // Colour the square the player is on
    this.colourTrail();

    // Check there are no walls blocking the path of the player
    if
(this.tiles[this.player.x/TILE_SIZE][this.player.y/TILE_SIZE].walls[1] == false &&

this.tiles[(this.player.x+TILE_SIZE)/TILE_SIZE][this.player.y/TILE_SIZE].walls[3] == false) {

        // Update the player's new x position
        this.player.newPositionX = this.player.x + TILE_SIZE;
        this.stepsTaken += 1;

    }
}
} else {
    // Invalid input
    // Raise invalid argument error here
    throw "Argument error. Direction specified must be 'up', 'down', 'left', 'right'";
}
}

// This method colours the tile beneath the player with the trail colour
this.colourTrail = function() {
    // Get x and y position of the player
    // Divide both the tile size so it can be used as an array index
    var playerXPosition = this.player.x / TILE_SIZE;
    var playerYPosition = this.player.y / TILE_SIZE;

    // Use two previously calculated values to set colour attribute of
    // the sprite beneath player to return value of player's trailColour
    // function
    this.tiles[playerXPosition][playerYPosition].colour =
        this.player.trailColour();
}

// Method generates a new maze
this.newMaze = function() {
    // Declare tiles as empty array
    var tiles = [];

    // Declare unvisitedTiles tiles as empty array
    var unvisitedTiles = [];

    // Iterate through y tiles up to maxY with counter y
    for (var x = 0; x < this.maxX; x++) {
        // Set each index in tiles to empty arrays
        tiles[x] = [];
        // Set each index in unvisitedTiles to empty arrays

```

```

unvisitedTiles[x] = [];

// Iterate through x tiles up to maxX with counter x
for (var y = 0; y < this.maxY; y++) {
    // Set the tile at index i,j to a 4 element array of zeroes
    tiles[x][y] = [0,0,0,0];
    // Set the same index in unvisitedTiles to true
    unvisitedTiles[x][y] = true;
}
}

// Determine random x and y positions in the maze
// Contstrain by values maxX and maxY
var randomX = Math.floor(Math.random()*this.maxY);
var randomY = Math.floor(Math.random()*this.maxX);
// Declare current cell as array with elements randomX and randomY
var currentTile = [randomX, randomY];

// Decalre path as array with one element - currentTile
var tileStack = [currentTile];

// Set the current tile as visited
unvisitedTiles[currentTile[0]][currentTile[1]] = false;

// Set visitedTiles equal to 1
var visitedTiles = 1;

// Loop through all available cell positions
while (visitedTiles < this.maxX * this.maxY) {
    // Determine potentialNeighboursential neighboring tiles
    var potentialNeighbours = [
        [currentTile[0]-1, currentTile[1], 0, 2],
        [currentTile[0], currentTile[1]+1, 1, 3],
        [currentTile[0]+1, currentTile[1], 2, 0],
        [currentTile[0], currentTile[1]-1, 3, 1]
    ];
    var neighbourTiles = [];

    // Determine if each neighboring cell is in game grid, and whether it has already been checked
    for (var l = 0; l < 4; l++) {
        if (potentialNeighbours[l][0] > -1 &&
            potentialNeighbours[l][0] < this.maxY &&
            potentialNeighbours[l][1] > -1 &&
            potentialNeighbours[l][1] < this.maxX &&
            unvisitedTiles[potentialNeighbours[l][0]][potentialNeighbours[l][1]]) {
            neighbourTiles.push(potentialNeighbours[l]);
        }
    }

    // If at least one active neighboring cell has been found
    if (neighbourTiles.length >= 1) {
        // Choose one of the neighbourTiles at random
        selectedTile = neighbourTiles[Math.floor(Math.random()*neighbourTiles.length)];

        // Remove the wall between the current cell and the chosen neighboring cell
    }
}

```

```

    tiles[currentTile[0]][currentTile[1]][selectedTile[2]] = 1;
    tiles[selectedTile[0]][selectedTile[1]][selectedTile[3]] = 1;

    // Mark the neighbor as visited, and set it as the current cell
    unvisitedTiles[selectedTile[0]][selectedTile[1]] = false;

    visitedTiles+=1
    currentTile = [selectedTile[0], selectedTile[1]];
    tileStack.push(currentTile);
}
// Otherwise go back up a step and keep going
else {
    currentTile = tileStack.pop();
}
}
return tiles;
}
}

```

**Engine.js**

```

/*
engine.js
Global scope file
-----
Written on the 04/01/2014 by Chris Jones
*/

// GLOBAL CONSTANTS
// These are immutable and do not change over course of program
//-----
const WIDTH      = 512;   // Width of the viewport
const HEIGHT     = 512;   // Height of the viewport
const TILE_SIZE = 32;    // The size of the tiles

const FPS = 30; // The fixed FPS (frames per second)
                // Controls the number of times screen is redrawn per second

const STATE = { // This is an object literal used as an 'enumeration'
    MENU: 0, // This is the state of the game
    MAZE: 1 // Used when it is necessary to check the current page
};

const BACKGROUND_COLOUR = "#333333"; // The colour of the background in hex
const WALL_COLOUR = "#222222"; // The colour of the walls in hex
const EXIT_COLOUR = "#ff8000"; // Colour of the exit in rgb
//-----

// INSTANTIATE GLOBAL VARIABLES
// These are implicitly defined to be global
// As they are in the global scope
//-----
// GAME is the object that wraps up the main functionality
// Methods and variables are added to this object later in the file
var GAME = {};

```

```
// Reference to the canvas element extracted from the HTML
var canvas = document.getElementById('canvas');

// Set the canvas dimensions equal to the constant dimensions
canvas.width = WIDTH;
canvas.height = HEIGHT;

// Reference to the drawing context of the canvas
var canvasContext = canvas.getContext('2d');

// The width of the maze initialised to 8
var mazeWidth = 8;

// The height of the maze initialised to 8
var mazeHeight = 8;

// Stores the level the player is on, starting at 1
var level = 1;

// Create a new maze object and store in variable maze
var maze = new Maze(
    mazeWidth,
    mazeHeight,
    canvasContext,
    new Camera()
);

var currentState = STATE.MENU;

// Initialise the newly instantiated maze
maze.initialise();
//-----
// NAVIGATION FUNCTIONS
//-----
// Instantiate the page stack as an empty array
// This will act as a stack of pages visited by the user
var pageStack = [];

function showPage(pageString) {
    // First hide all of the pages
    $('.page').css('display', 'none');

    // Then display the desired page
    $(pageString).css('display', 'inline-flex');
}

// Navigate to page with HTML id pageString
function navigateTo(pageString) {
    if (pageString == ".main-menu") {
        // If the currently stored level is one
        if (localStorage["level"] == 1) {
            // Disable the continue button
```

```

$('.btn-continue').attr('disabled', 'disabled');
// Set the text within the button to 'Continue'
// This is incase it contains a level number
$('.btn-continue').html('Continue');

} else {
    // Otherwise insert the level the player is currently on
    $('.bt n-continue').removeAttr('disabled');
    $('.btn-continue').html(
        'Continue <br><p class="under-text">From level ' +
        localStorage["level"] + "</p>"
    );
}
}

// If navigating away from the canvas
if (pageStack[pageStack.length-1] == "#canvas") {
    // Disable the player
    maze.player.disabled = true;
    // Switch the current state to menu
    currentState = STATE.MENU;
}

// If navigating to the canvas
if (pageString == "#canvas") {
    // Enable the player
    maze.player.disabled = false;

    // Switch the current state to game
    currentState = STATE.MAZE;
}

// Show the page that is being navigated to
showPage(pageString);

// Push it to the top of the pageStack
pageStack.push(pageString);
}

// Go back to the last visited page
function goBack() {
    // Check there are at least two pages in the stack
    // Otherwise there is no menu to navigate back to
    if (pageStack.length >= 2) {
        // Pop the top page off of the stack.
        // This has the effect of navigating back a page
        pageStack.pop();

        // Navigate to the new top of the stack
        // Note this is popped off here
        // and pushed back on in navigation
        navigateTo(pageStack.pop());
    }
}
//-----

```

```
// DEFINE THE FUNCTIONS / VARIABLES THAT CONTROL THE GAME
// These are bound to the object GAME
// This is instantiated above
//-----
// This function handles saving the state of the GAME to localStorage
GAME.save = function() {
    // Store the value of level in key "level" of localStorage
    localStorage["level"] = level;

    // Store the value of mazeWidth in the key "x" of localStorage
    localStorage["x"] = mazeWidth;

    // Store the value of mazeHeight in the key "y" of localStorage
    localStorage["y"] = mazeHeight;
}

// This function handles loading GAME state from localStorage
GAME.load = function() {
    // Retrieve the width of the maze stored
    mazeWidth = parseInt(localStorage["x"]);

    // Retrieve the height of the maze store
    mazeHeight = parseInt(localStorage["y"]);

    // Retrieve the level that is stored
    level = parseInt(localStorage["level"]);
}

// Function handles all of the update logic necessary in the game
GAME.update = function() {
    // Update the logic of the maze
    maze.update();

    // Check if maze is complete
    if (maze.complete) {

        $('#time').html(
            (String(parseInt(maze.completedSeconds / 60)).length > 1 ?
            parseInt(maze.completedSeconds / 60) : "0" +
            parseInt(maze.completedSeconds / 60)) + ":" +
            (String(maze.completedSeconds).length > 1 ? maze.completedSeconds % 60 : "0" +
            maze.completedSeconds % 60));

        $('#steps').html(String(maze.stepsTaken));

        // If so, generate a new maze with two more cells in each axis
        maze = new Maze(mazeWidth+=2, mazeHeight+=2, canvasContext, new Camera());
        // Initialise this new maze
        maze.initialise();
    }

    // Display the end of level menu
    navigateTo('.end-of-level-menu');
}
```

```
// Increment level  
level +=1;  
  
// Save the value of the maze  
this.save();  
}  
}  
  
// Draw the maze and associated sprites  
GAME.draw = function() {  
  
// Fill in the background colour  
canvasContext.fillStyle = BACKGROUND_COLOUR;  
canvasContext.fillRect(0,0,WIDTH,HEIGHT);  
  
// Draw the maze to the canvas  
maze.draw();  
}  
  
// Event listener function that deals with input  
GAME.doKeyDown = function(e) {  
// Swtich statement must on keyCode variable of the event  
switch(e.keyCode) {  
    // Up arrow or W key codes  
    case 87:  
    case 38:  
        // Update player position by moving it up  
        maze.updatePlayerPosition("up");  
        break;  
    // Down arrow or S key codes  
    case 83:  
    case 40:  
        // Update player position by moving it down  
        maze.updatePlayerPosition("down");  
        break;  
  
    // Down arrow or S key codes  
    case 65:  
    case 37:  
        // Update player position by moving it left  
        maze.updatePlayerPosition("left");  
        break;  
  
    // Right arrow or D key codes  
    case 68:  
    case 39:  
        // Update player position by moving it right  
        maze.updatePlayerPosition("right");  
        break;  
  
    // Escape key code  
    case 27:  
        // If the current page is not the pause menu  
        // and the current page is the game canvas  
        if (pageStack[pageStack.length-1] != ".pause-menu" &&
```

```

pageStack[pageStack.length-1] == "#canvas") {

    // Navigate to the pause menu
    navigateTo('.pause-menu');

} else {

    // Go back a step in the stack of pages
    goBack();

}

}

//-----
// INPUT HANDLER
//-----
// Add a keyboard event listener
window.addEventListener( "keydown", GAME.doKeyDown, false);
//-----

// INITIALISE THE MAIN GAME
//-----
// setInterval is a function that takes a function as parameter.
// Takes time interval in milliseconds as second parameter
// Calls the function passed first every period that
// was passed as second parameter
setInterval(function() { // Anonymous function ...
    // Check the game is in maze state before calling functions
    if (currentState == STATE.MAZE) {
        // Call the update method first
        GAME.update();

        // Call the draw method
        GAME.draw();
    }

    //console.log(maze.completedSeconds);

}, 1000/FPS);

// Navigate to the main menu
navigateTo('.main-menu');
//-----


// EVENTS FOR BUTTON CLICKS
//-----
// Continue button click event
$('.btn-continue').click(function() {
    // Load the game state data from storage
    GAME.load();
}

```

```
// Instantiate a new maze with respect to the loaded data
maze = new Maze(
    mazeWidth,
    mazeHeight,
    canvasContext,
    new Camera()
);

// Initialise the new maze
maze.initialise();

// Navigate to the canvas game page
navigateTo('#canvas');

// New game button click event
$('.btn-new-game').click(function() {
    // Set level to 1
    level = 1;

    // Set maze dimensions to 4
    mazeWidth = 4;
    mazeHeight = 4;

    // Instantiate a new maze
    maze = new Maze(
        mazeWidth,
        mazeHeight,
        canvasContext,
        new Camera()
    );

    // Initialise the new maze
    maze.initialise();

    // Save the game, at level one
    // This ensures the continue button is enabled
    GAME.save();

    // Navigate to the game
    navigateTo('#canvas');
});

// Help button click event
$('.btn-help').click(function() {
    // Navigate to the first help page
    navigateTo('.help-1');
});

// Navigate back a page when back button is pressed
$('.btn-back').click(function() {
    // Check if the top of the pageStack is a help page
    // Back behavior is different for help pages
    if (pageStack[pageStack.length-1].indexOf('help') > -1) {
```

```
// While there are still help pages in the stack
while (pageStack[pageStack.length-1].indexOf('help') > -1) {
    // Pop this page off the stack
    pageStack.pop();
}

// Navigate to top of the stack
navigateTo(pageStack.pop());

// Otherwise go back a page
} else {
    goBack();
}
});

// Handle the left arrow button event
$('.left').click(function() {
    // Get the index of the current help page
    var currentPageIndex =
        pageStack[pageStack.length-1][pageStack[pageStack.length-1].length-1];

    // Subtract one from currentPageIndex
    var nextPage = parseInt(currentPageIndex) - 1;

    // Navigate to the previous help page in the sequence
    navigateTo('.help-' + nextPage);
});

// Handle the right arrow button event
$('.right').click(function() {
    // Get the index of the current help page
    var currentPageIndex =
        pageStack[pageStack.length-1][pageStack[pageStack.length-1].length-1];

    // Add one to currentPageIndex
    var nextPage = parseInt(currentPageIndex) + 1;

    // Navigate to the next help page in the sequence
    navigateTo('.help-' + nextPage);
});

// Handle the next level button on the end of level menu
$('.btn-next-level').click(function() {
    // Show the canvas game page
    goBack();
});

// Handle the quit button event
$('.btn-quit').click(function() {
    // Jump straight to the root menu
    // Empty the whole page stack
    // Do this by popping off all elements
    // while the array still has elements
    while (pageStack.length != 0) {
        pageStack.pop();
    }
});
```

```
}

// Save the game
GAME.save();

// Navigate to the main menu
navigateTo('.main-menu');
});

//-----
```

# Testing

## Introduction

This section pertains to the thorough testing of every aspect of the new system that has been developed, along with evidence of the results of these tests presented. The end of this section will provide the client with documents that present each stage of testing. Following a discussion based interview, the client will determine whether the testing shows sufficient rigour to prove the system functions as was intended. If the client believes that testing is indeed sufficient then she will sign it off.

## Alpha test evidence

Below is a copy of the alpha test plan table presented in the *Test Strategy* section. References to headers of evidence of alpha testing for each of the software areas in the table has been added as evidence that sufficient alpha testing has taken place in the development of the new system.

Software area to be tested	Description of alpha test	Headers for alpha testing in Software Development
HTML webpage	Ensure that all the content is displayed on the webpage.	<b>Alpha testing the HTML</b>
CSS styling	Ensure the appropriate styling is applied to the webpage.	<b>Alpha testing the CSS</b>
Camera class	Run a test using the JavaScript console in a web browser in order to determine that an object of type Camera can be instantiated correctly.	<b>Camera class function alpha testing</b>
Sprite class	Run a test using the JavaScript console in a web browser in order to determine that an object of type Sprite can be instantiated correctly.	<b>Alpha testing the Sprite function</b>
SpriteBatch class	<p>This will require the writing of additional test code that will be temporarily inserted into the JavaScript file and removed after this alpha test is complete.</p> <p>Run an initial test to ensure that a SpriteBatch object can draw rectangle sprites by drawing a grid of sprites.</p> <p>Run another test to ensure that a SpriteBatch object can draw walls by adding walls to the grid of sprites.</p>	<b>Initial Alpha testing of the SpriteBatch, Second alpha test of the draw function</b>
Timer class	Write some test code in the Timer JavaScript file that will instantiate and run a one second timer that outputs the number of elapsed second to the JavaScript console of a web browser. Inspect whether these updates appear every second.	<b>Alpha testing Timer</b>
Player class	Copy and modify the test code that was used for SpriteBatch in order to draw a player onto the grid of sprites. Test whether the player moves correctly and its colour fades correctly.	<b>Alpha testing the player class function</b>
ExitTile class	Run a test using the JavaScript console in a web browser in order to determine that an object of type ExitTile can be instantiated correctly.	<b>Alpha testing exit tile</b>
Maze class	<p>Test that a maze can be drawn and updated.</p> <p>Test that the random generation of mazes produces solvable mazes.</p> <p>Test that player moves and updates trail correctly.</p>	<b>First alpha test of Maze , Alpha testing the Maze generation</b>

Menus / Completed program	Test that each of the menus is displayed correctly and all of the menu elements function correctly (e.g. help button navigates to help menu 1).	<b>Alpha Testing Engine.js file</b>
---------------------------	---	-------------------------------------

## Post development testing

### Introduction

This is a crucial stage in the development of the new system. With the system developed and alpha testing having already taken place, the post development testing will allow each area of the software to be thoroughly tested. The *Test Strategy*, earlier in the report, contains details on each of the tests that need to take place to ensure the system is functional and error-free. The *Test Strategy* will be used to organise the testing and evaluate the system at this stage.

### Format of testing

The testing stage is laid out much like that of the *Test Strategy*, with each test corresponding to one defined in the *Test Strategy* section of the report. There are three tables – input tests, processing tests and output tests. Below is a dissection of the columns in these three tables.

- **Test No** – This is a number that corresponds with a number in the *Test Strategy* section. This represents a reference to a test in the test strategy. This allows for quick lookup of any given test for further information, so that the complete details of the test need not be reproduced in this section.
- **Objective Reference(s)** – Each test has one or more references to objectives in the *Final Design Objectives* section in the *Nature of the solution*. These are the objectives that the test will evaluate for success. Each design objective has at least one test associated with it. This ensures that the whole system will be rigorously tested.
- **Purpose of test** - This is copied from the test strategy and explains the reason for the test.
- **Input / Test data** – This is copied from the test strategy and contains any data or inputs that are entered into the system in order to produce the conditions required for the test. Note all tests assume that the webpage containing the game is visited. This field may contain N/A, which means no test data need be entered in order to test the related aspect of the system.
- **Expected Result** – This is copied from the test strategy and is the result that is expected from the system in order for the test to pass.
- **Actual Result (evidence)** – The actual result of the testing on the system. Note these are contained as images organised into reference tables.
- **Test successful?** – A yes or no, which indicates whether the test was a success or failure.

### Testing References

Each test's actual result contains a reference to the testing image references. These are images organised into three tables (input, process and output testing image references) that represent the evidence of the actual results of the testing. Note that where this evidence is not self-explanatory the images there may be additional text or labelling of the images.

**Test table**

<b>Test No</b>	<b>Objective Reference(s)</b>	<b>Purpose of test</b>	<b>Input / Test data</b>	<b>Expected Result</b>	<b>Actual Result (evidence)</b>	<b>Test successful?</b>
1	1.a.i	Test that pressing the new game button on the main menu will start a new maze.	New game button	A new maze is displayed.	Reference 1	Yes
2	1.a.ii	Test that pressing the continue button, when disabled, does nothing.	Continue button (must be disabled)	Nothing.	Reference 2	Yes
3	1.a.ii	Test that pressing the continue button, when enabled, loads a level from local storage and displays it.	Continue button	A maze is displayed that contains the width and height dimensions of the ones specified in the game's local storage.	Reference 3	Yes
4	1.a.ii	Test that, when game data is saved in local storage, that pressing the continue button loads a maze of the correct size.	Data in local storage. Continue game button.	The maze generated and displayed when the continue button is pressed will match the dimensions saved in local storage.	Reference 4	Yes
5	1.a.iii	Test that pressing the help button on the main menu will navigate to the first help menu.	Help button	The first help menu is displayed.	Reference 5	Yes
6	1.a.iv	Test that pressing the back button on the help menus navigates to the main menu.	Back button (first, second and third help menus respectively)	For each of three back buttons, main menu is displayed.	Reference 6	Yes
7	1.a.v	Test that pressing the right arrow button on the first help menu navigates to the second help menu.	Right arrow button (on first help menu)	The second help menu is displayed.	Reference 7	Yes
8	1.a.v	Test that pressing the right arrow button on the second help menu navigates to the third help menu.	Right arrow button (on second help menu)	The third help menu is displayed.	Reference 8	Yes
9	1.a.v	Test that pressing on the disabled right arrow button on the third help menu does nothing.	Right arrow button (on third help menu)	Nothing.	Reference 9	Yes
10	1.a.vi	Test that pressing on the left arrow	Left arrow button (on third help	The second help menu is displayed.	Reference	Yes

		button on the third help menu navigates to the second help menu.	menu)		10	
11	1.a.vi	Test that pressing on the left arrow button on the second help menu navigates to the first help menu.	Left arrow button (on second help menu)	The first help menu is displayed.	Reference 11	Yes
12	1.a.vi	Test that pressing on the disabled left arrow button on the first help menu does nothing.	Left arrow button (on first help menu)	Nothing	Reference 12	Yes
13	1.a.vii	Test that when the quit to main menu on the end of level menu is pressed that it navigates to the main menu	Quit to main menu button (on end of level menu)	The main menu is displayed.	Reference 13	Yes
14	1.a.vii	Test that pressing the quit to main menu button on the pause menu navigates to the main menu	Quit to main menu button (on pause menu)	Main menu is displayed.	Reference 14	Yes
15	1.a.viii	Test that when the next level button on the end of level menu is pressed that it displays a new maze.	Next level button	A new maze is displayed.	Reference 15	Yes
16	1.b.i	Test that W key moves player up by one tile if not blocked by a wall	W key	Player moves up one tile.	Reference 16	Yes
17	1.b.ii	Test that Up arrow key moves player up by one tile if not blocked by a wall	Up arrow key	Player moves up one tile.	Reference 17	Yes
18	1.b.iii	Test that A key moves player left by one tile if not blocked by a wall	A key	Player moves left one tile.	Reference 18	Yes
19	1.b.iv	Test that Left arrow key moves player left by one tile if not blocked by a wall	Left arrow key	Player moves left one tile.	Reference 19	Yes
20	1.b.v	Test that S key moves player down by one tile if not blocked by a wall	S key	Player moves down one tile.	Reference 20	Yes
21	1.b.vi	Test that Down arrow key moves player down by one tile if not blocked by a wall	Down arrow key	Player moves down one tile.	Reference 21	Yes
22	1.b.vii	Test that D key moves player right by one tile if not blocked by a wall	D key	Player moves right one tile.	Reference 22	Yes
23	1.b.viii	Test that Right key moves player right by	Right arrow key	Player moves right one tile.	Reference	Yes

		one tile if not blocked by a wall			23	
24	1.b. ix	Test that pressing the escape key when a maze is displayed navigates to the pause menu	Escape key (when maze is displayed)	The pause menu is displayed.	Reference 24	Yes
25	1.b. ix	Test that pressing the escape key when on the pause menu returns to displaying the maze	Escape key (on pause menu)	Maze is displayed.	Reference 25	Yes
26	1.b. ix	Test that pressing the escape key when on the main menu does nothing.	Escape key (on main menu)	Nothing.	Reference 26	Yes
27	1.b. ix	Test that pressing the escape key when on a help menu does nothing	Escape key	Nothing.	Reference 27	No
28	2.a.i, 2.d.iv	Move the player to the exit tile of a 4x4 maze and ensure the maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Combination of arrow keys in order to reach exit tile.	The exit tile should be reachable and thus confirm the maze is solvable.	Reference 28	Yes
29	2.a.i, 2.d.iv	Check that the 6x6 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Combination of arrow keys in order to reach exit tile.	The exit tile should be reachable and thus confirm the maze is solvable.	Reference 29	Yes
30	2.a.i, 2.d.iv	Check that the 8x8 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Combination of arrow keys in order to reach exit tile.	The exit tile should be reachable and thus confirm the maze is solvable.	Reference 30	Yes
31	2.a.i, 2.d.iv	Check that the 10x10 maze is solvable by ensuring that moving the player to the exit tile completes the maze.	Combination of arrow keys in order to reach exit tile.	The exit tile should be reachable and thus confirm the maze is solvable.	Reference 31	Yes
32	2.a.ii	Test that a 4 by 4 tile maze is generated when the new game button on the main menu is pressed.	New game button on main menu.	A maze is generated and displayed that is 4 by 4 tiles in size (16 total tiles).	Reference 32	Yes
33	2.a.iii	Test that the exit tile and player are placed randomly in the maze at least a third of the maze apart from each other. Repeat this for multiple random mazes.	New game button.	The player and exit tile are randomly placed in the maze, at least a third of the size of the maze apart.	Reference 33	Yes
34	2.a.iv,	Test that, for a maze larger than the	<i>Input required reaching suitable</i>	The camera is centred on the player and	Reference	Yes

	3.l.xii	dimensions of the game window (512 by 512 pixels), then the camera is centred on the player and follows the player, after panning has occurred.	<i>level larger than game window.</i>  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player).	follows the player when the player moves.	34	
35	2.a.v	Test that, for a maze larger than the dimensions of the game window (512 by 512 pixels), the camera begins the level centred on the exit tile then pans over to the player.	<i>Input required reaching suitable level larger than game window.</i>	The camera should pan from the exit tile to the player just after starting maze that cannot fit wholly in the game window.	Reference 35	Yes
36	2.b	Test that every subsequent maze that is generated is 2 tiles larger in the horizontal and vertical dimensions.	New game button  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to complete maze).  Next level button.  <i>Last two sets of input are repeated for 4 levels.</i>	Subsequent mazes that are generated should be 2 tiles larger in the horizontal and vertical dimensions than the previous level.	Reference 36	Yes
37	2.c	Test that, if a game is saved in local storage and local storage is deleted, the game disables the continue button	Delete the contents of local storage in the JavaScript console.  Visit the webpage.	When local storage is deleted the continue button should be disabled.	Reference 37	<b>No</b>
38	2.c	Test that, when quitting to the main menu from the next level menu, that the continue button is enabled and the game can be loaded	Quit to main menu button (on end of level menu).  Continue button.	The continue button should not be disabled and when pressed should generate and display a maze of dimensions saved in local storage.	Reference 38	<b>No</b>
39	2.c	Test that, when quitting to the main	Quit to main menu button (on	The continue button should not be disabled	Reference	<b>No</b>

		menu from the pause menu, that the continue button is enabled and the game can be loaded.	end of level menu). Continue button.	and when pressed should generate and display a maze of dimensions saved in local storage.	39	
40	2.d.i	Test that pressing W key does not move player up by one tile when the path is blocked by a wall	W key	Player stays on the same tile.	Reference 40	Yes
41	2.d.i	Test that pressing A key does not move player left by one tile when the path is blocked by a wall	A key	Player stays on the same tile.	Reference 41	Yes
42	2.d.i	Test that pressing S key does not move player down by one tile when the path is blocked by a wall	S key	Player stays on the same tile.	Reference 42	Yes
43	2.d.i	Test that pressing D key does not move player right by one tile when the path is blocked by a wall	D key	Player stays on the same tile.	Reference 43	Yes
44	2.d.i	Test that pressing Up arrow key does not move player up by one tile when the path is blocked by a wall	Up arrow key	Player stays on the same tile.	Reference 44	Yes
45	2.d.i	Test that pressing Left arrow key does not move player left by one tile when the path is blocked by a wall	Left arrow key	Player stays on the same tile.	Reference 45	Yes
46	2.d.i	Test that pressing Down arrow key does not move player down by one tile when the path is blocked by a wall	Down arrow key	Player stays on the same tile.	Reference 46	Yes
47	2.d.i	Test that pressing Right arrow key does not move player right by one tile when the path is blocked by a wall	Right arrow key	Player stays on the same tile.	Reference 47	Yes
48	2.d.ii	Test that pressing W key does not move player up by one tile when the player is on the top edge of the maze.	W key	Player stays on the same tile.	Reference 48	Yes
49	2.d.ii	Test that pressing A key does not move player left by one tile when the player is	A key	Player stays on the same tile.	Reference 49	Yes

		on the left edge of the maze.				
50	2.d.ii	Test that pressing S key does not move player down by one tile when the player is on the bottom edge of the maze.	S key	Player stays on the same tile.	Reference 50	Yes
51	2.d.ii	Test that pressing D key does not move player right by one tile when the player is on the right edge of the maze.	D key	Player stays on the same tile.	Reference 51	Yes
52	2.d.ii	Test that pressing Up arrow key does not move player up by one tile when the player is on the top edge of the maze.	Up arrow key	Player stays on the same tile.	Reference 52	Yes
53	2.d.ii	Test that pressing Left arrow key does not move player left by one tile when the player is on the left edge of the maze.	Left arrow key	Player stays on the same tile.	Reference 53	Yes
54	2.d.ii	Test that pressing Down arrow key does not move player down by one tile when the player is on the bottom edge of the maze.	Down arrow key	Player stays on the same tile.	Reference 54	Yes
55	2.d.ii	Test that pressing Right arrow key does not move player right by one tile when the player is on the right edge of the maze.	Right arrow key	Player stays on the same tile.	Reference 55	Yes
56	2.d.iii	Check that pressing the W, A, S, D, Up arrow, Right arrow, Down arrow and Left arrow keys when on the main menu does not cause the player to move tiles in the maze.	Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow.  New game button.	Upon displaying of maze the player has not performed any movement.	Reference 56	Yes
57	2.d.iii	Check that pressing the W, A, S, D, Up arrow, Right arrow, Down arrow and Left arrow keys when on the pause menu does not cause the player to move tiles in the maze.	Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow and right arrow.  Escape key.	Upon displaying of maze the player has not performed any movement.	Reference 57	Yes
58	2.d.iii	Check that pressing the W, A, S, D, Up	Arbitrary combination of W, A, S,	Upon displaying of maze the player has not	Reference	Yes

		arrow, Right arrow, Down arrow and Left arrow keys when on the end of level menu does not cause the player to move tiles in the maze.	D, up arrow, left arrow, down arrow and right arrow.  Next level button.	performed any movement.	58	
59	2.d.v, 3.l.xiv	Test that the player leaves a trail correctly.	New game button  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player).	The player should leave a trail on all the tiles that it moves over.	Reference 59	Yes
60	2.d.vi, 3.l.xiii	Test that the colour of the player fades correctly over time. The player should fade from green to purple and then back again in a recurring cycle.	New game button.	The player should fade from green to purple and then back again in a recurring cycle.	Reference 60	Yes
61	3.a	Test that the game is embedded in a single web page.	N/A	The game should be embedded in a single webpage	Reference 61	Yes
62	3.b.i, 3.b.ii, 3.b.iii, 3.b.iv, 3.l.i	Test to ensure that there is a heading, subheading, canvas and footer present on the webpage.	N/A	The web page should contain: <ul style="list-style-type: none"><li>• A header with the name of the game (Mazr).</li><li>• A subheading that explains the game briefly.</li><li>• The game should be embedded in a HTML canvas.</li><li>• A footer that contains a link to the user documentation.</li></ul>	Reference 62	Yes
63	3.c	Test that when JavaScript is disabled in the user's browser, then a suitable error message is displayed and none of the menus are displayed.	N/A	A suitable error message should be displayed when the user has JavaScript turned off in their browser. No menus should be displayed.	Reference 63	No
64	3.d	Test that the game will be composed of a set of menus and each will be contained in div HTML elements and all appear on	N/A	The game should be composed of a set of menus and each will be contained within a HTML div element. Only one menu should be	Reference 64	Yes

		the same page although hidden / unhidden to simulate multiple pages being navigated.		visible at a given time.		
65	3.e.i, 3.e.ii, 3.e.iii	Test that the main menu contains correct content.	N/A	<p>The main menu should contain:</p> <ul style="list-style-type: none"> <li>• A button to continue with a game saved in local storage. This will be displayed if no game is stored in local storage.</li> <li>• A button to start a new game.</li> <li>• A button to go to the first help menu.</li> </ul>	Reference 65	Yes
66	3.f.i, 3.f.ii, 3.f.iii, 3.g.i, 3.g.ii	Test that the first help menu contains the correct content.	N/A	<p>All help menus (and thus the first help menu) should contain:</p> <ul style="list-style-type: none"> <li>• A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.</li> <li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li> <li>• A back button to return to the main menu from any of the help menus.</li> </ul> <p>The first help menu should contain:</p> <ul style="list-style-type: none"> <li>• Onscreen help that explains the format of the game and the goal of the game.</li> <li>• An image of a small maze with a player and exit tile.</li> </ul>	Reference 66	Yes
67	3.f.i, 3.f.ii, 3.f.iii, 3.h.i,	Test that the second help menu contains the correct content.	N/A	<p>All help menus (and thus the second help menu) should contain:</p> <ul style="list-style-type: none"> <li>• A right arrow button, which can be used to navigate to the following help menu.</li> </ul>	Reference 67	Yes

	3.h.ii			<p>If there is no following help menu then this button will be disabled.</p> <ul style="list-style-type: none"> <li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li> <li>• A back button to return to the main menu from any of the help menus.</li> </ul> <p>The second help menu should contain:</p> <ul style="list-style-type: none"> <li>• Onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.</li> <li>• Image of a small maze where the player is leaving a coloured trail moving to the exit tile.</li> <li>• </li> </ul>		
68	3.f.i, 3.f.ii, 3.f.iii, 3.i.i, 3.i.ii	Test that the third help menu contains the correct content.	N/A	<p>All help menus (and thus the third help menu) should contain:</p> <ul style="list-style-type: none"> <li>• A right arrow button, which can be used to navigate to the following help menu. If there is no following help menu then this button will be disabled.</li> <li>• A left arrow button, which can be used to navigate to the previous help menu. If there is no previous help menu then this button will be disabled.</li> <li>• A back button to return to the main menu from any of the help menus.</li> </ul> <p>The third help menu should contain:</p> <ul style="list-style-type: none"> <li>• Onscreen help that explains how</li> </ul>	Reference 68	Yes

				<p>completing a maze subsequently generates a larger maze for the player to solve.</p> <ul style="list-style-type: none"> <li>• Image of a larger maze.</li> </ul>		
69	3.j.i, 3.j.ii, 3.j.iii, 3.j.iv, 3.j.v	Test that the end of level menu contains the correct content.	N/A	<p>The end of level menu should contain:</p> <ul style="list-style-type: none"> <li>• Text congratulating the user on their completion of the maze.</li> <li>• Text that contains the time taken for the user to complete the maze.</li> <li>• Text that contains the number of steps taken for the user to complete the maze.</li> <li>• A button that allows the user to go to the next maze.</li> <li>• A button that allows the user to quit to the main menu.</li> </ul>	Reference 69	Yes
70	3.k.i, 3.k.ii	Test that the pause menu contains the correct content.	N/A	<p>The pause menu should contain:</p> <ul style="list-style-type: none"> <li>• A button that allows the user to quit to the main menu.</li> </ul>	Reference 70	Yes
71	3.l.ii	Check that the game canvas is 512 by 512 pixels	New game button.	The screen should be 512px by 512px.	Reference 71	Yes
72	3.l.iii	Test the maze is drawn to the canvas as a grid of tiles with walls, along with an exit tile and a player.	New game button	A grid of tiles that represent the maze must be drawn to the canvas, along with the exit tile and player.	Reference 72	Yes
73	3.l.iv	Check that the tiles in the maze are 32 by 32 pixels	New game button.	The tiles in the maze should be 32px by 32px.	Reference 73	Yes
74	3.l.v	Check that the player in the maze is 32 by 32 pixels	New game button.	The player should be 32px by 32px.	Reference 74	Yes
75	3.l.vi	Check that the exit tile in the maze is 32 by 32 pixels	New game button.	The exit tile should be 32px by 32px.	Reference 75	Yes

76	3.l.vii	Check that the walls in the maze are 32 by 2 pixels	New game button.	The walls of the maze should be 32 by 2 pixels.	Reference 76	Yes
77	3.l.viii	Test that The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of grey.	New game button  <i>Input required to reach 6 different levels in order to check shades of grey.</i>  Arbitrary combination of W, A, S, D, up arrow, left arrow, down arrow, right arrow (to move player) and next level button repeatedly.	The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of grey.	Reference 77	Yes
78	3.l.ix	Test that the saturation and lightness values of the player remain constant at 80% and 40% respectively, to the closest integer.	New game button	The player should have a constant saturation of 80% and a constant lightness of 40%, to the closest integer.	Reference 78	Yes
79	3.l.x	Test that the colour of the exit tile is #FF8000	New game button	The exit tile should have the colour #FF8000	Reference 79	Yes
80	3.l.xi	Test that, for a maze smaller than the dimensions of the game window (512 by 512 pixels), then the maze is fixed in the centre of the window and the camera is disabled.	New game button.	When the maze does not fit in the game window (512 by 512 pixels) then the camera should be enabled and follow the player around the maze.	Reference 80	Yes

## Testing References

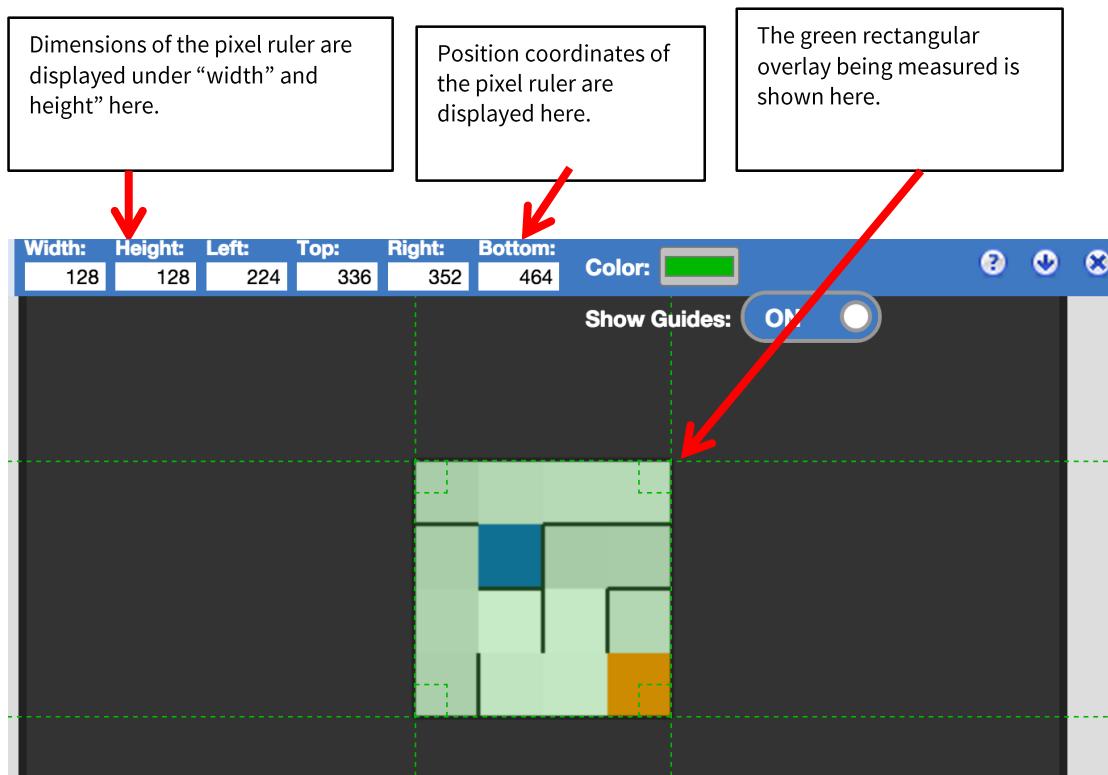
All of the references to the tests in the previous test table are included in the table below. Each of the test references has a unique number associated with it that corresponds to a test in the test table. References contain primarily screenshots of the system along with textual explanations of the test results. Some more contain screenshots of specific tools being used to test a facility of the game.

Note that this table includes evidence of the results of the tests in the testing section and does not include evidence of the procedure required to produce the result. Consult the *Test Strategy* test plan for a method to reproduce the results in each of the tests below.

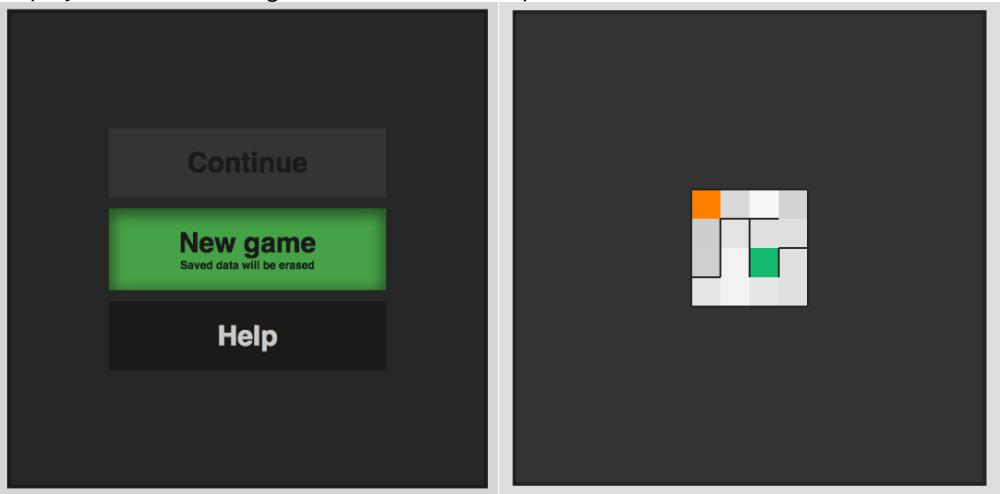
### Note on the pixel ruler

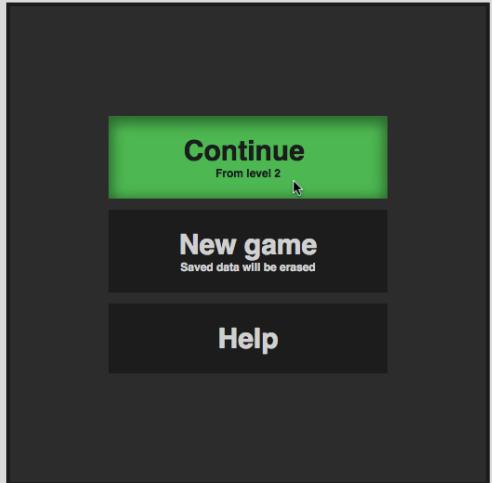
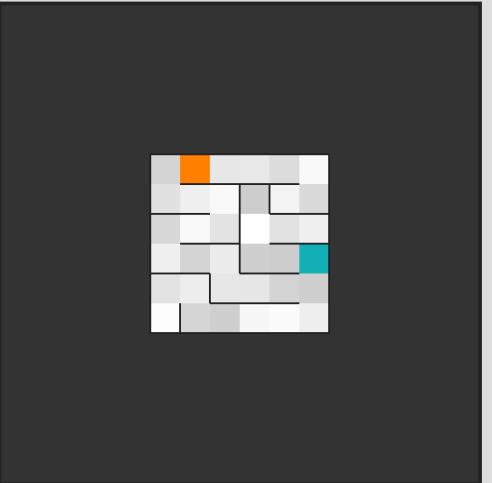
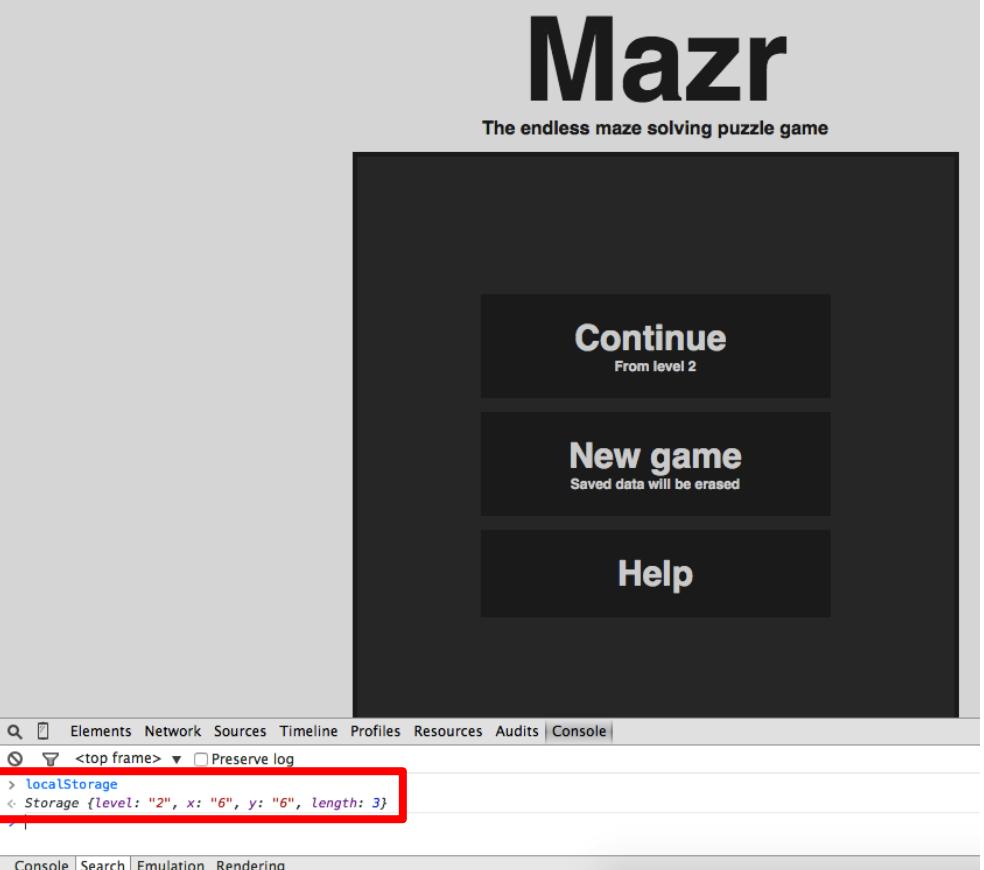
Many of the tests that follow include the use of a pixel ruler. This is used to measure pixel distances on the webpage when it is rendered. This method is perfect for any test that requires the size of a game element to be measured (e.g. a tile or a wall) or something related to the game geometry (e.g. does the player stay centred on the centre of the screen when the camera is enabled).

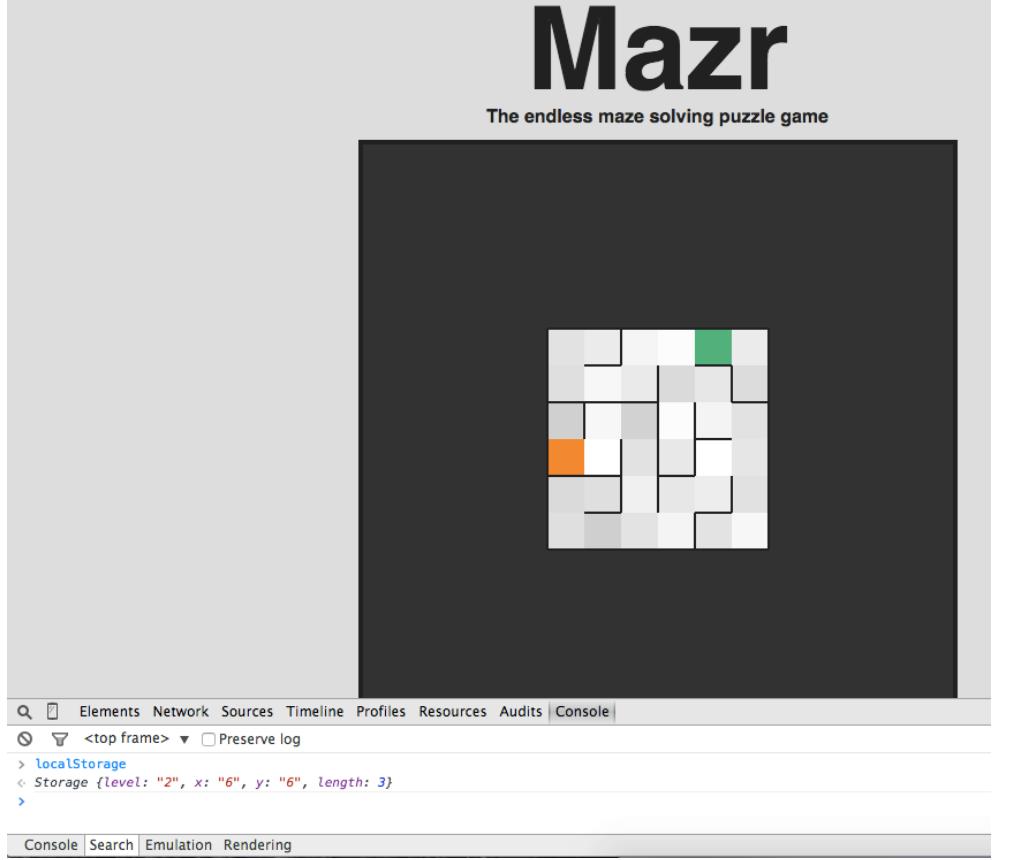
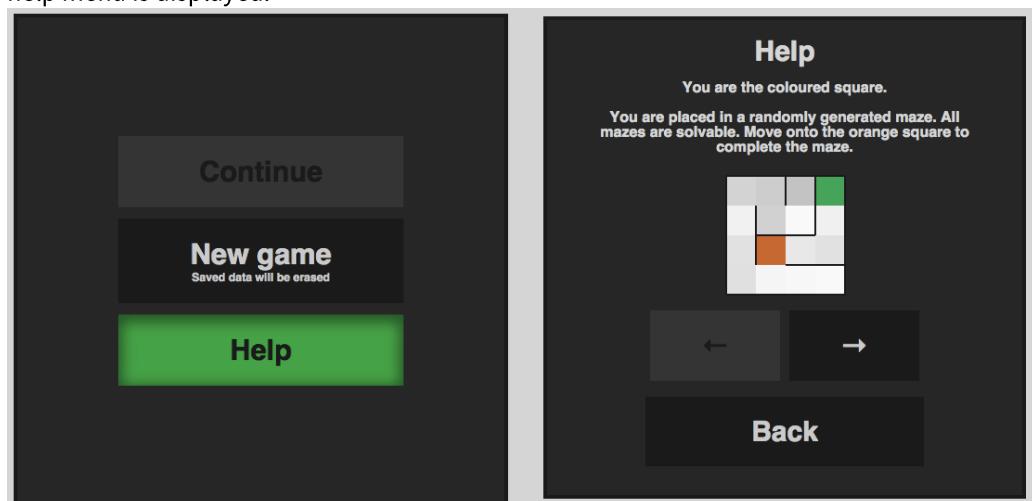
An explanation of how the pixel ruler is used in these tests is shown below. When enabled, a bar will appear at the top of a window that describes the width and height of a green overlay rectangle that can be dragged over the webpage. This is in essence the pixel ruler and its position and dimensions are displayed in the top bar. This system can be used to measure different elements of the game by dragging the green rectangle over them or otherwise positioning it to give meaningful geometrical information on a test.

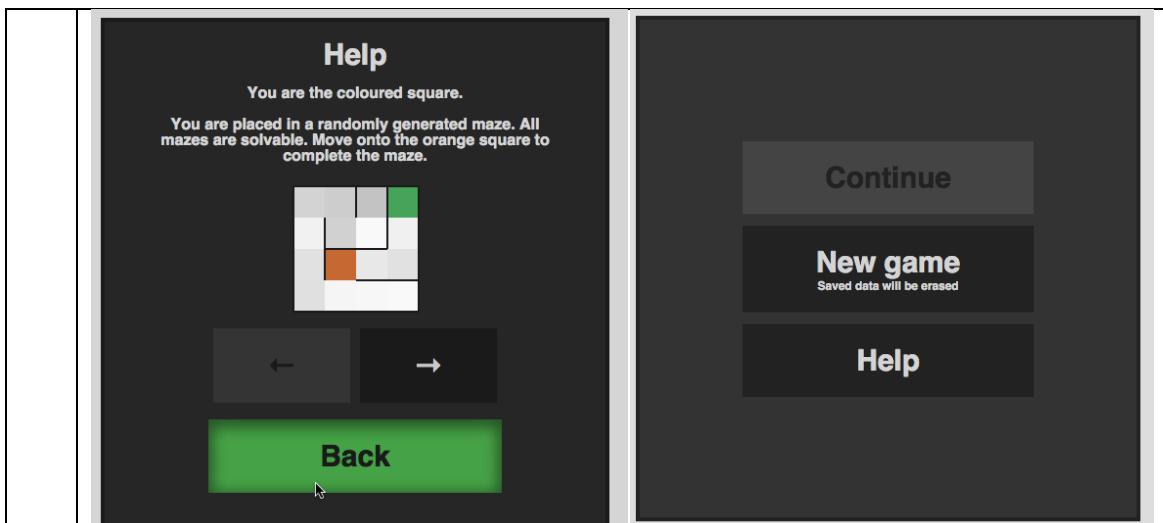


**Reference table**

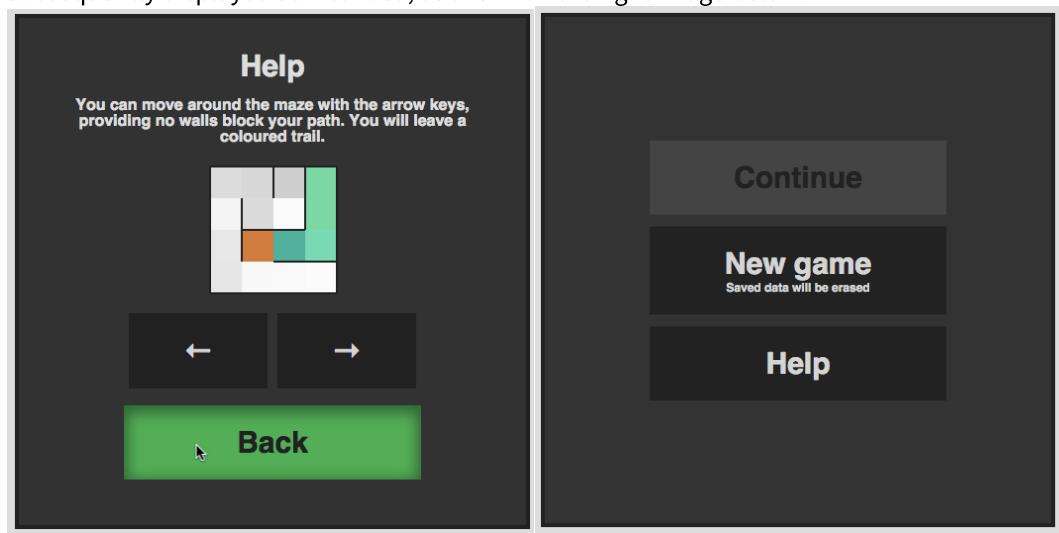
Ref No	Reference contents
1	<p>Left image shows new game button being pressed. Right image below shows a new maze is displayed once the new game button has been pressed.</p> 
2	<p>Image below shows that when the disabled continue button is pressed then nothing occurs.</p> 
3	<p>Left image below shows the continue button being pressed to load a level 2 maze. Level 2 maze should have <math>(4+2) = 6</math> tiles in each dimension to be correctly loaded. Right image below shows that a maze is displayed. By visual inspection it can be seen the maze is 6 by 6 tiles, which is correct for level 2.</p>

	 
4	<p>First image below shows in the JavaScript console that there is a level 2, 6 by 6 tile maze saved in Local Storage – shown highlighted in a red rectangle.</p>  <p>Second image below shows result of pressing the continue button. A maze is displayed that, by visually inspection, can be determined as 6 by 6 tiles in size. This is correct as it matches the values stored in Local Storage.</p>

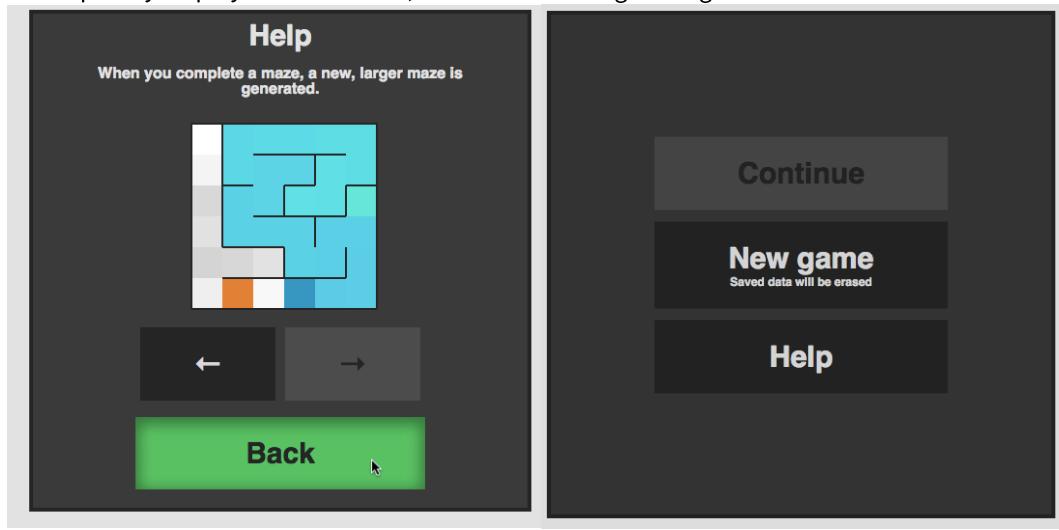
	
5	<p>Left image below shows the help button being pressed. Left image below shows that the first help menu is displayed.</p> 
6	<p>In the left image below the back button is pressed on first help menu and the Main menu is subsequently displayed as intended, as shown in the right image below.</p>



In the left image below the back button is pressed on second help menu and the Main menu is subsequently displayed as intended, as shown in the right image below.

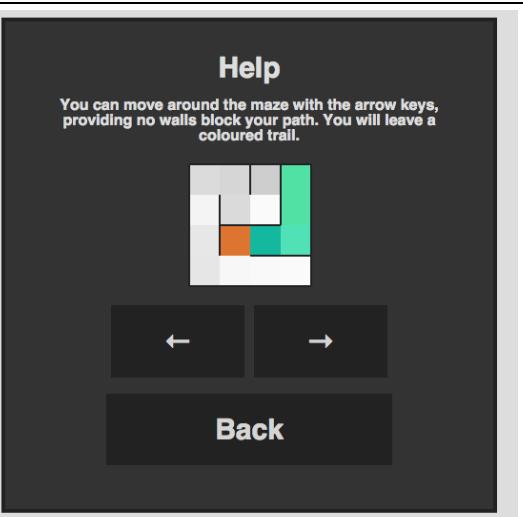
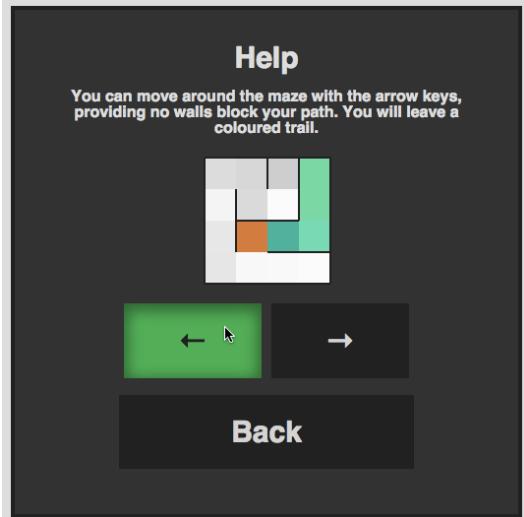
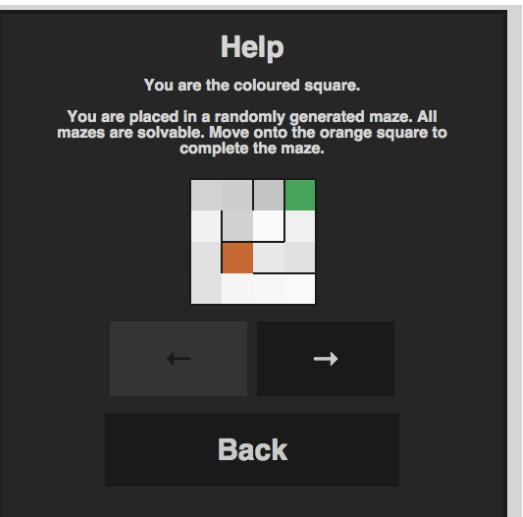
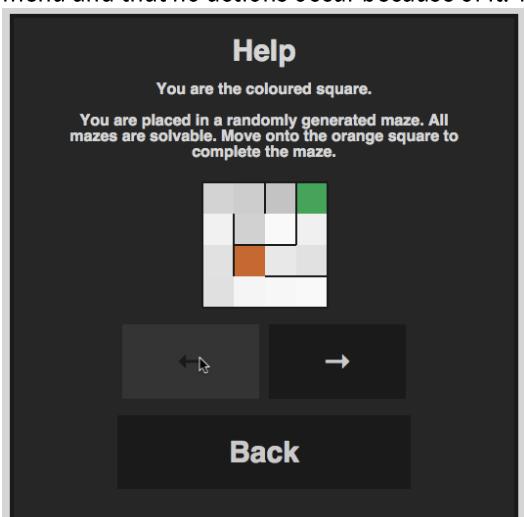


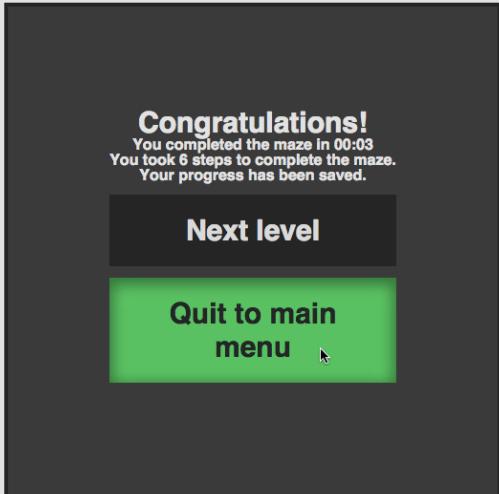
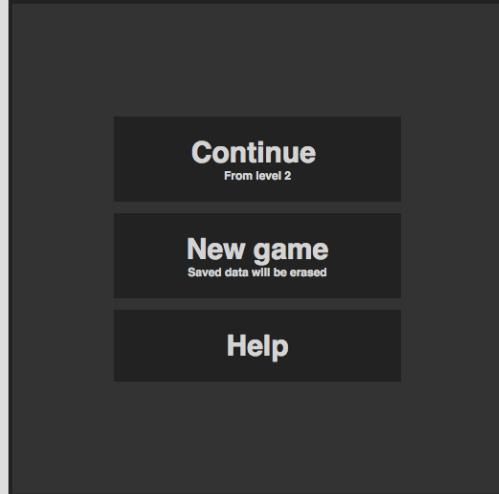
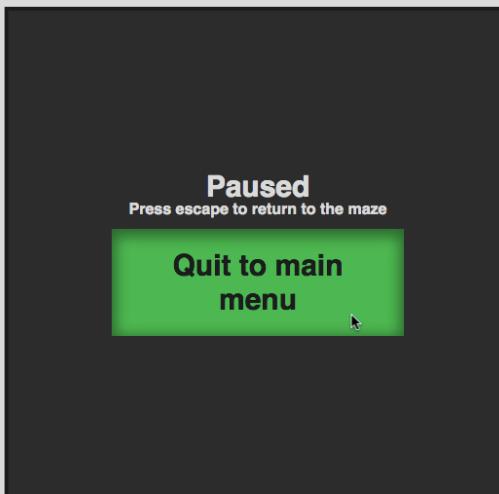
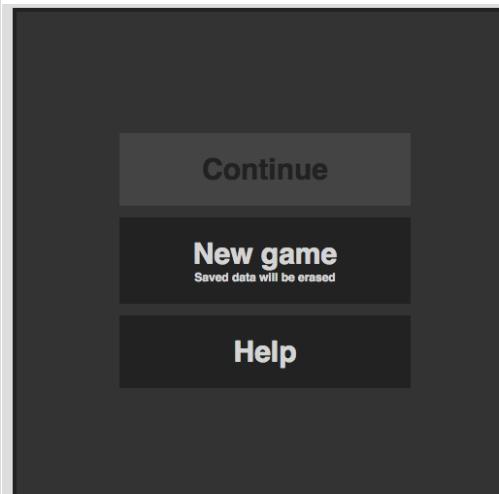
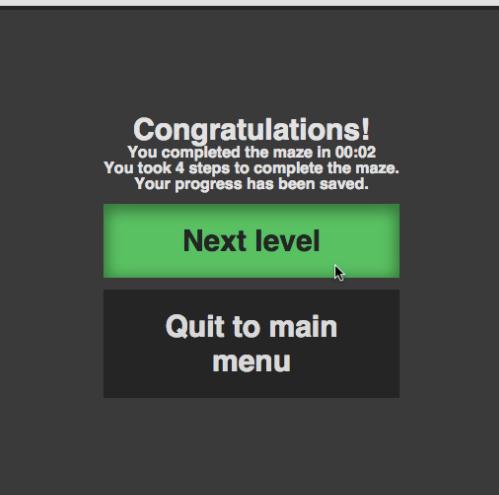
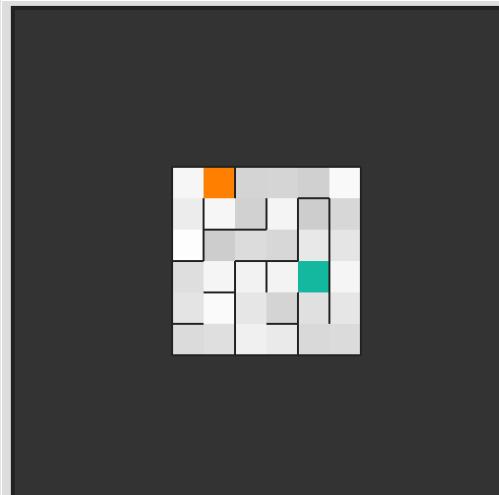
In the left image below the back button is pressed on third help menu and the Main menu is subsequently displayed as intended, as shown in the right image below.

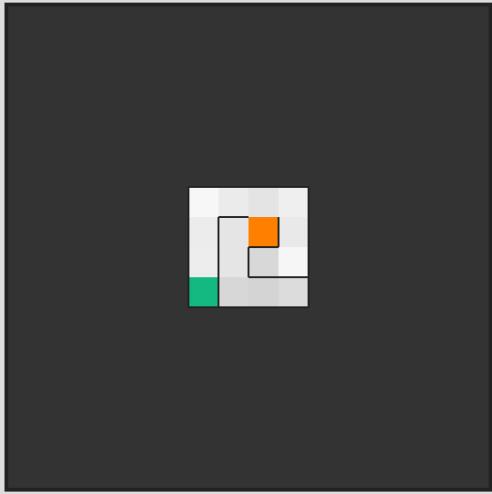
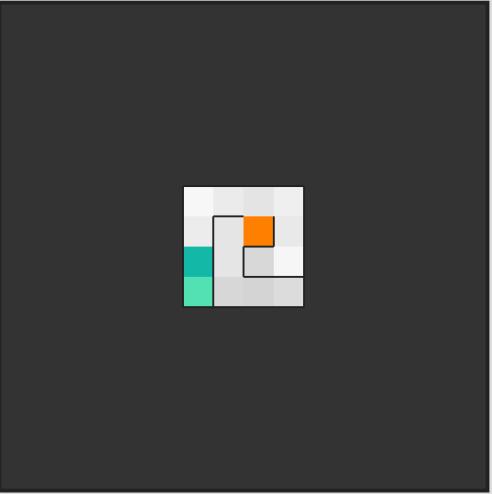
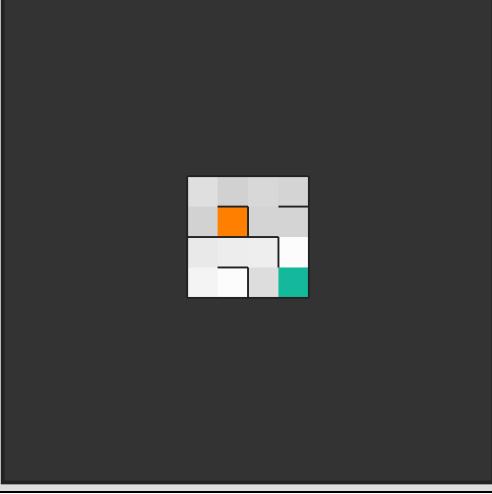
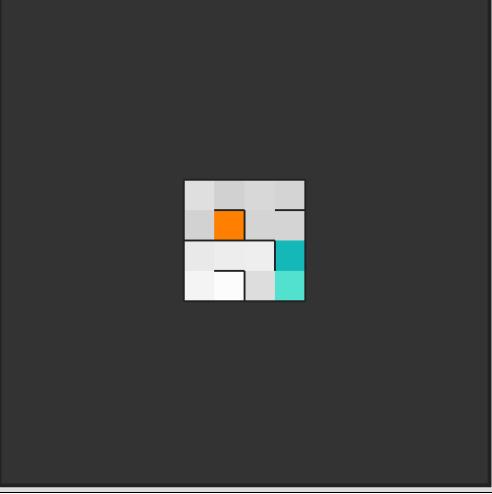
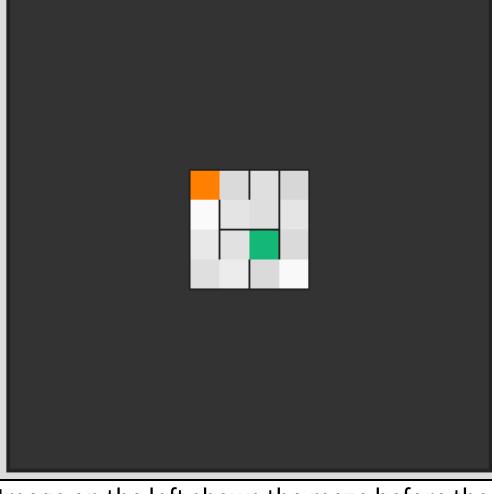
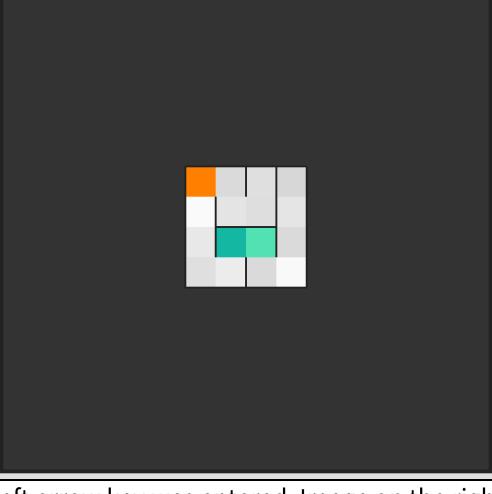


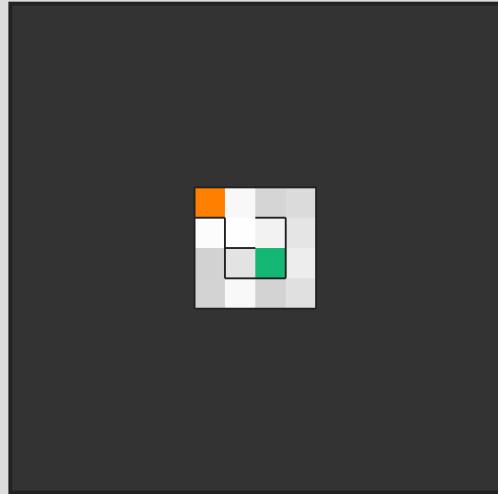
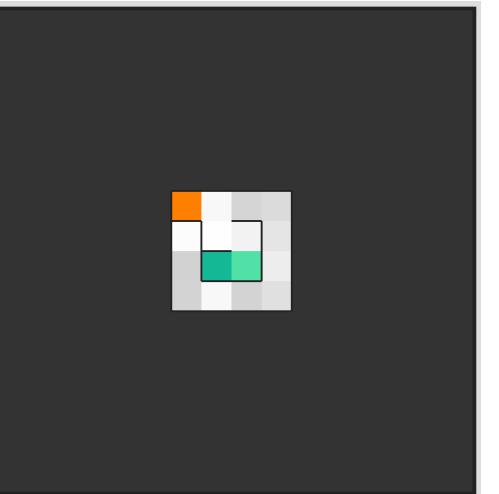
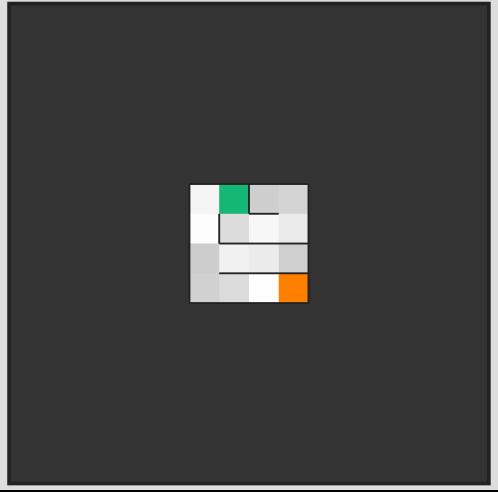
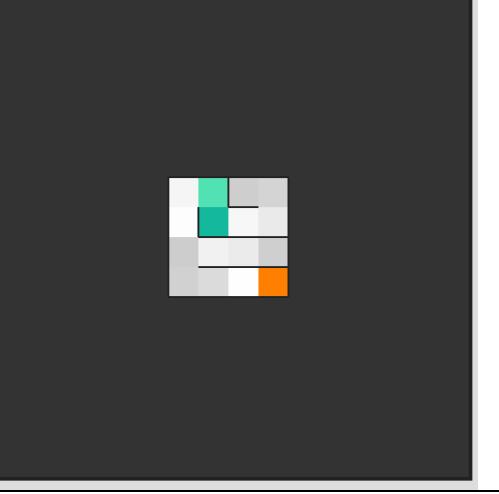
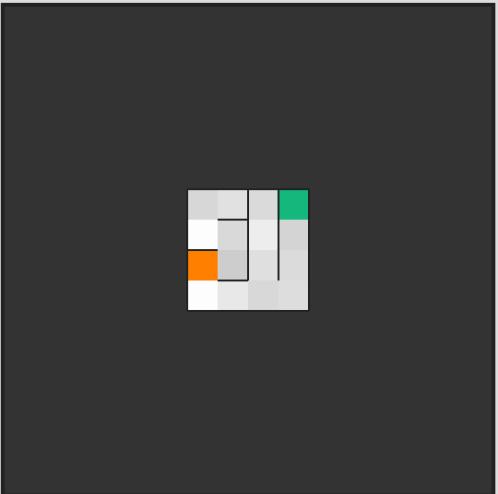
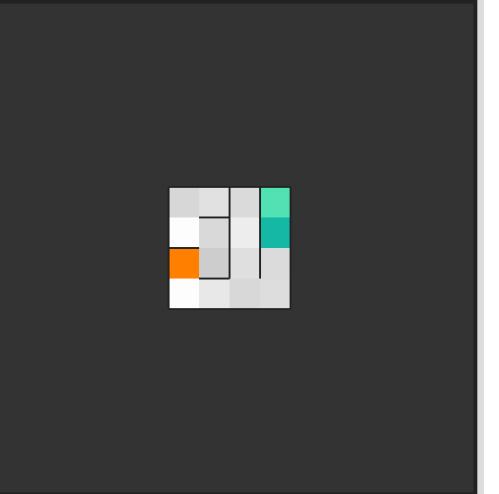
7 Left image below shows the first help menu with the right arrow button being pressed. As

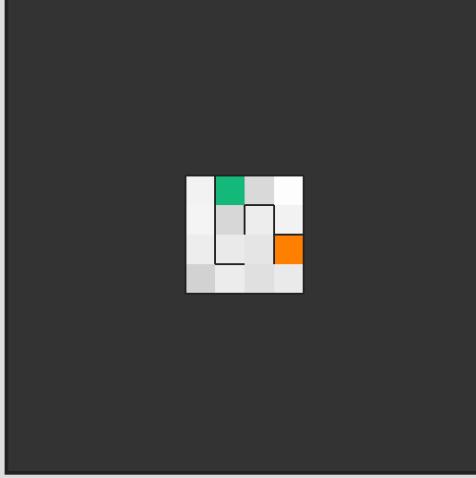
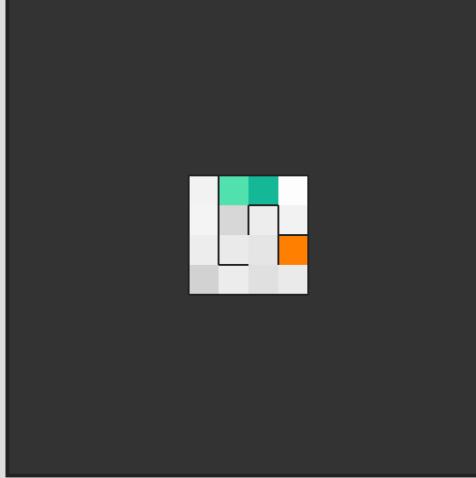
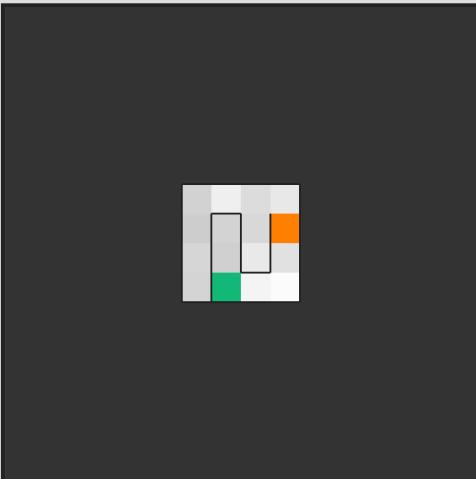
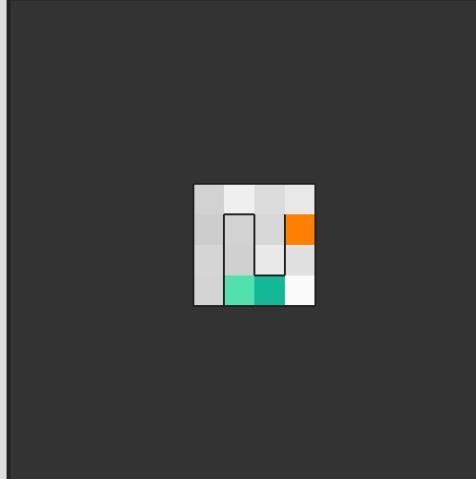
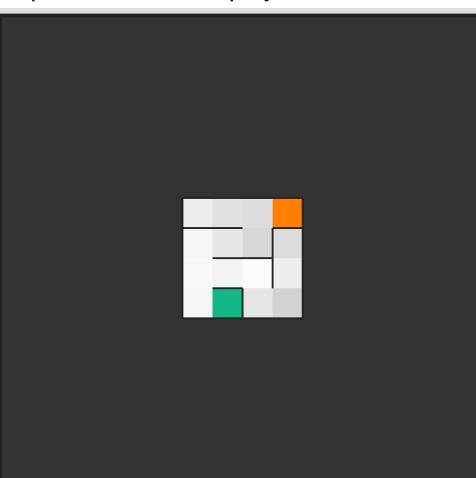
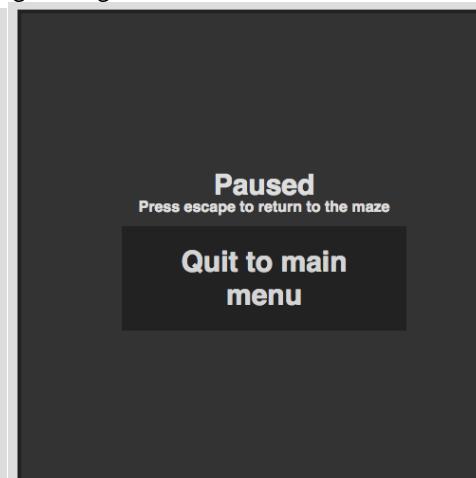
	<p>shown in the right image below, the second help menu is displayed, as intended.</p>
8	<p>The right arrow button on the second help menu is pressed, as shown in the left image below. Subsequently the third help menu is displayed in the right image below, as intended.</p>
9	<p>The image below shows that nothing helps when the disabled right arrow button is pressed on the third help menu. This is as intended.</p>
10	<p>The left image below shows the left arrow button being pressed on the third help menu. Subsequently, as shown in the right image below, the second help menu is displayed. This is as intended.</p>

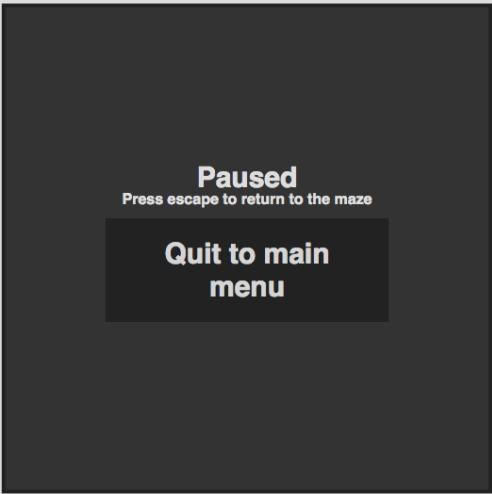
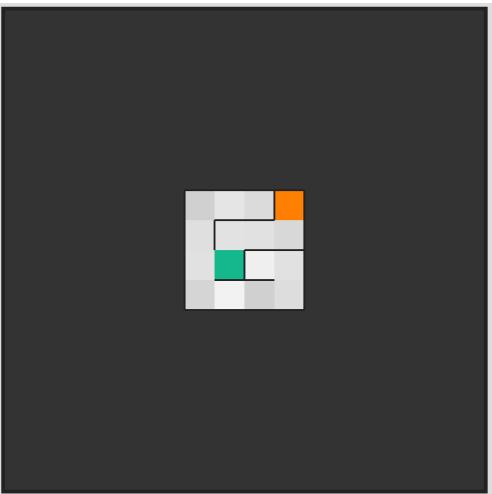
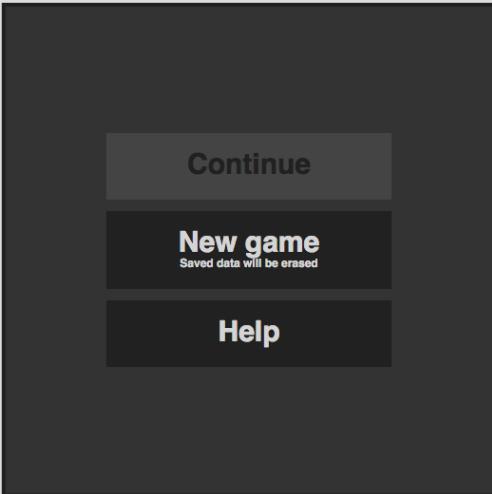
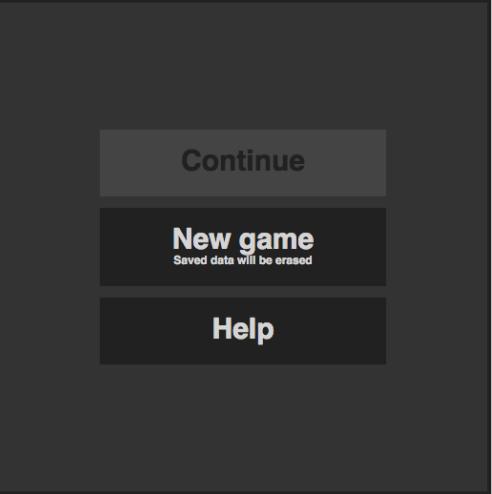
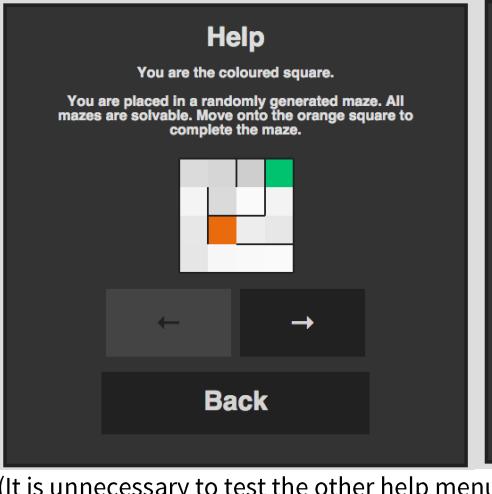
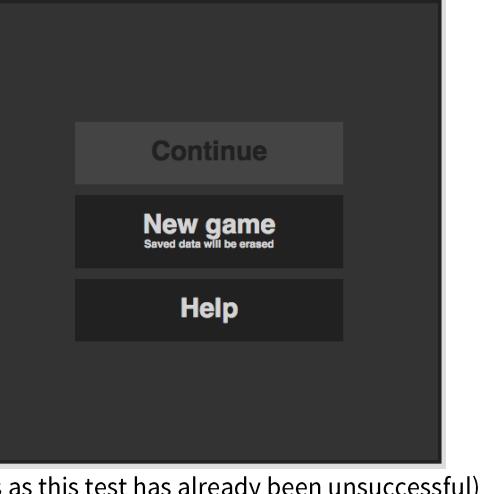
		
11	The left image below shows the left arrow button being pressed on the second help menu. Subsequently, as shown in the right image below, the first help menu is displayed. This functions as intended.	 
12	Below is an image that shows the disabled left arrow button being pressed on the first help menu and that no actions occur because of it. This is as intended.	
13	The left image below shows the Quit to main menu button on the end of level menu being pressed. This causes the main menu to subsequently be displayed, as shown in the right image below. This therefore functions as intended.	

		
14	The left image below shows the Quit to main menu button being pressed on the End of level menu. The right image shown below shows subsequently that the main menu is displayed. This functions as intended.	 
15	Left image below shows the Next level button being pressed on the End of level menu. Subsequently, as shown in the right image below, a new maze is displayed. This functions as intended.	 
16	Image on the left shows the maze before the W key was entered. Image on the right shows maze, along with the correct movement of the player up by one tile.	

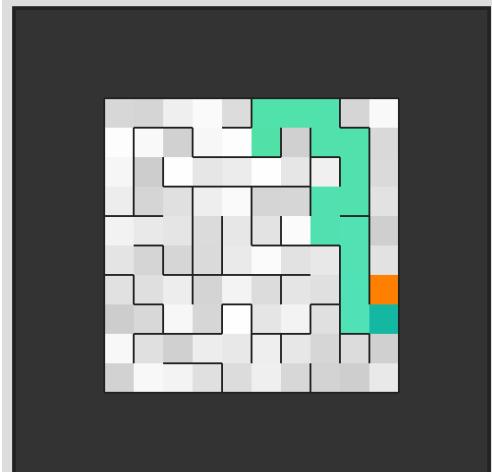
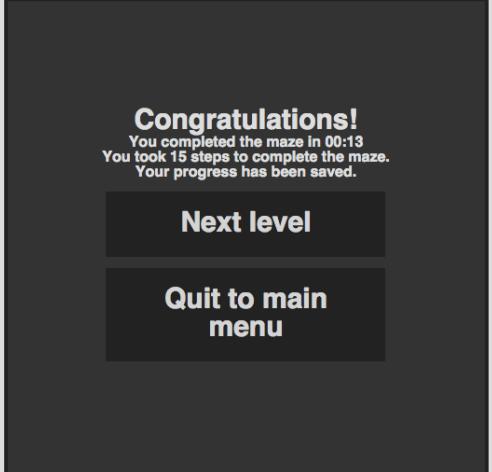
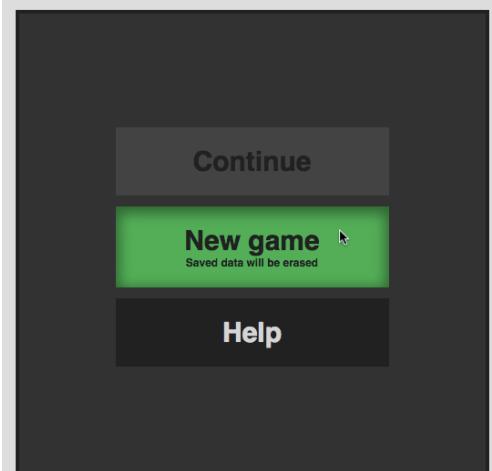
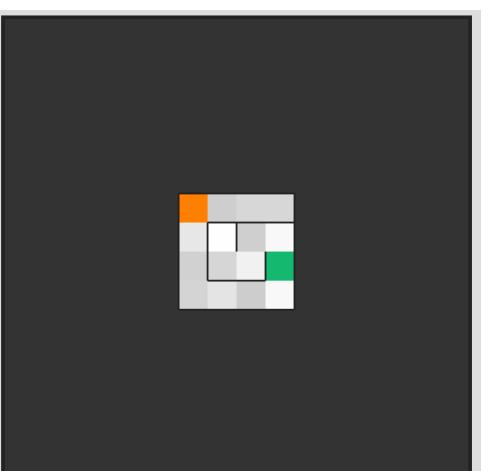
			
17	Image on the left shows the maze before the Up arrow key was entered. Image on the right shows maze, along with the correct movement of the player up by one tile.		
18	Image on the left shows the maze before the A key was entered. Image on the right shows maze, along with the correct movement of the player left by one tile.		
19	Image on the left shows the maze before the Left arrow key was entered. Image on the right shows maze, along with the correct movement of the player left by one tile.		

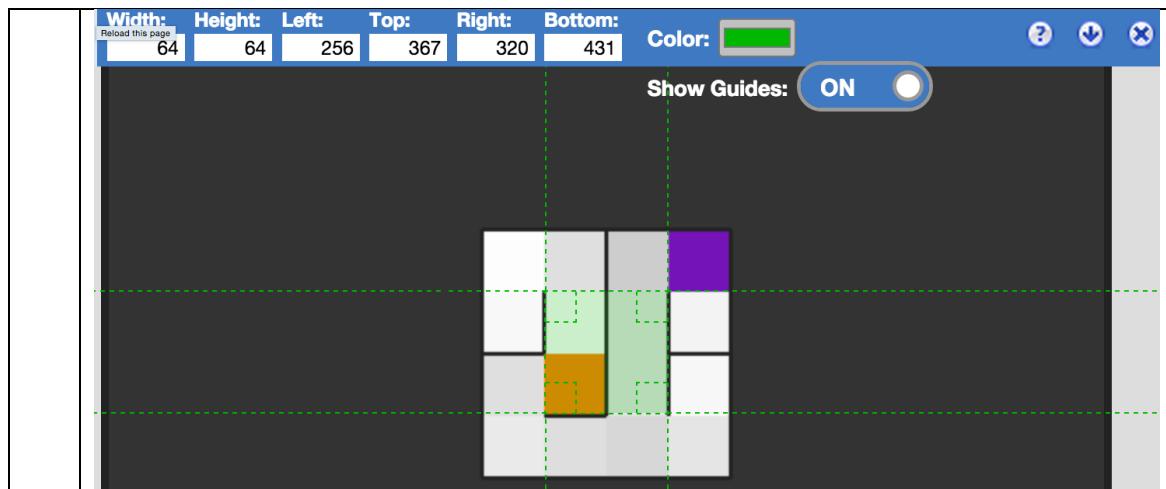
	 A 5x5 grid representing a maze. The player character (a green square) is at position (2, 3). There is an orange wall at (1, 2), a grey wall at (2, 1), and a white space at (3, 2).	 The same 5x5 grid. The player has moved down one tile to position (3, 3). The orange wall is now at (1, 3), and the grey wall is at (2, 2).	
20	Image on the left shows the maze before the S key was entered. Image on the right shows maze, along with the correct movement of the player down by one tile.	 A 5x5 grid representing a maze. The player character (a green square) is at position (3, 2). There is an orange wall at (2, 1), a grey wall at (3, 1), and a white space at (4, 2).	 The same 5x5 grid. The player has moved down one tile to position (4, 2). The orange wall is now at (3, 1), and the grey wall is at (4, 1).
21	Image on the left shows the maze before the Down arrow key was entered. Image on the right shows maze, along with the correct movement of the player down by one tile.	 A 5x5 grid representing a maze. The player character (a green square) is at position (3, 3). There is an orange wall at (1, 3), a grey wall at (2, 3), and a white space at (4, 3).	 The same 5x5 grid. The player has moved right one tile to position (4, 3). The orange wall is now at (1, 4), and the grey wall is at (2, 4).
22	Image on the left shows the maze before the D key was entered. Image on the right shows maze, along with the correct movement of the player right by one tile.		

	 A 4x4 grid representing a maze. The player character (a green square) is at position (1,1). There is a wall (orange square) at (1,2), a door (green square) at (2,1), and a goal (orange square) at (4,2).	 The same 4x4 grid as the left image, but the player character has moved to position (2,1), indicating a rightward movement.	
23	Image on the left shows the maze before the Right arrow key was entered. Image on the right shows maze, along with the correct movement of the player right by one tile.		
	 A 4x4 grid representing a maze. The player character (a green square) is at position (1,1). There is a wall (orange square) at (1,2), a door (green square) at (2,1), and a goal (orange square) at (4,2).	 The same 4x4 grid as the left image, but a pause menu is overlaid. It displays the word "Paused" and the instruction "Press escape to return to the maze". Below that is a button labeled "Quit to main menu".	
24	The left image shows a maze that is being displayed. Upon the escape key being pressed, then the pause menu is displayed as shown in the right image. This is as intended.		
	 A 4x4 grid representing a maze. The player character (a green square) is at position (1,1). There is a wall (orange square) at (1,2), a door (green square) at (2,1), and a goal (orange square) at (4,2).	 The same 4x4 grid as the left image, but a pause menu is overlaid. It displays the word "Paused" and the instruction "Press escape to return to the maze". Below that is a button labeled "Quit to main menu".	
25	The left image shows the pause menu being displayed. Upon the escape key being pressed, then the pause menu is no longer displayed and a maze is displayed, as shown in the right image. This is as intended.		

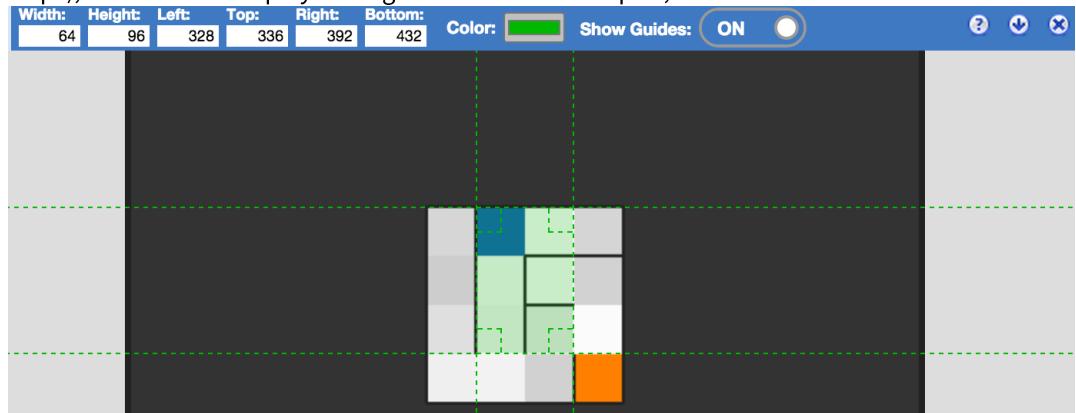
	 A screenshot of a game paused. A black rectangular overlay covers the top half of the screen. Inside, the word "Paused" is centered in white, followed by the instruction "Press escape to return to the maze". At the bottom of this overlay is a white button labeled "Quit to main menu" in black text.  A screenshot of a 4x4 maze grid. The player character is a green square at the bottom-left. The exit tile is an orange square at the top-right. All other tiles are grey, forming a standard L-shaped maze path.
26	Image on left shows that the main menu is displayed. Upon the escape key being pressed, the main menu continues to be displayed as in the right image and no other actions occur. This is as intended.  A screenshot of a main menu. It features three buttons: "Continue" (top), "New game" (middle, with the sub-instruction "Saved data will be erased" below it), and "Help" (bottom).  A screenshot showing the same main menu as the left image, but with the "Continue" button highlighted in a darker shade of grey, indicating it has been selected.
27	The left image shows the first help menu. Upon the escape key being pressed, the main menu is displayed, as shown in the second image on the right. This is undesirable behaviour and results in an unsuccessful test.  A screenshot of a help menu titled "Help". It contains text: "You are the coloured square.", "You are placed in a randomly generated maze. All mazes are solvable. Move onto the orange square to complete the maze.", and a small 4x4 maze diagram with a green start square, an orange exit square, and grey walls. Below the diagram are two arrows: a left arrow on the left and a right arrow on the right. At the bottom is a "Back" button.  A screenshot of the main menu, identical to the one in test 26, with the "Continue" button highlighted.
	(It is unnecessary to test the other help menus as this test has already been unsuccessful)
28	Left image below shows a 4 by 4 maze, with the player moved to one tile away from the exit tile. Right image below shows that once the player is moved up by another tile with input of the W key then the maze is solved and the end of level menu is displayed. This shows that the maze is solvable and the exit tile is reachable, as intended.

29	Left image below shows a 4 by 4 maze, with the player moved to one tile away from the exit tile. Right image below shows that once the player is moved left by one tile with the input of the D key then the maze is solved and the end of level menu is displayed. This shows the maze is solvable and the exit tile is reachable, as intended.	
30	Left image below shows an 8 by 8 maze, with the player moved to one tile away from the exit tile. Right image below shows that once the player is moved up by one tile with the input of the W key then the maze is solved and the end of level menu is displayed. This shows the maze is solvable and the exit tile is reachable, as intended.	
31	Left image below shows an 10 by 10 maze, with the player moved to one tile away from the exit tile. Right image below shows that once the player is moved up by one tile with the input of the W key then the maze is solved and the end of level menu is displayed. This shows the maze is	

	<p>solvable and the exit tile is reachable, as intended.</p>  
32	<p>Left image shows the New game button being pressed. Right image shows that a new maze is generated and displayed subsequently. This is as intended.</p>  
33	<p>This test involves the use of a pixel ruler in order to measure the distance apart of the player and exit tile. Note that the horizontal and vertical distances are measured by using a rectangular overlay that has a labelled width and height. Two opposite corners of the rectangle are placed in the same corner of the two different tiles (for example, the rectangle's top left and bottom right corners might be lined up with the top left corner of BOTH the exit tile and the player). This denotes the pixel distance vertically and horizontally by the height and width made by the rectangle overlay respectively.</p> <p>Multiple mazes of three different sizes (the first three levels) will be sampled in this manner. A calculation is then performed for each maze to determine whether the player and exit tile are greater than one third apart.</p> <p>In the first image below, the width and height of the maze in pixels is <math>32 \times 4 = 128</math> pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 64 pixels in either direction, is greater than a third of 128 pixels (approx. 43 px), the exit tile and player are greater than a third apart, as intended.</p>



In the second image below, the width and height of the maze in pixels is  $32 \times 4 = 128$  pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 64 pixels in the horizontal and 96 pixels in the vertical, is greater than a third of 128 pixels (approx. 43 px), the exit tile and player are greater than a third apart, as intended.



In the third image below, the width and height of the maze in pixels is  $32 \times 6 = 192$  pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 128 pixels in the horizontal and 96 pixels in the vertical, is greater than a third of 128 pixels (64 px), the exit tile and player are greater than a third apart, as intended.

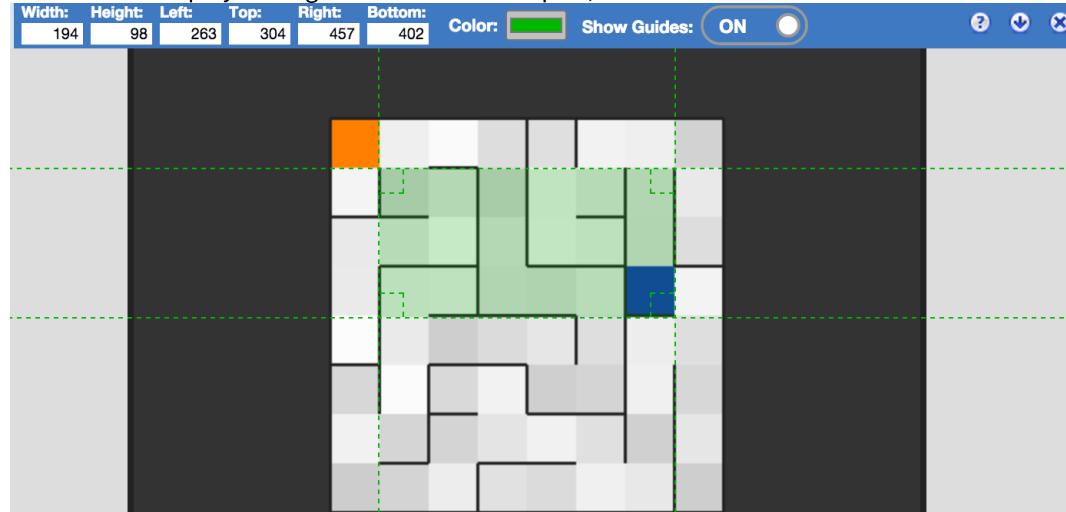


In the fourth image below, the width and height of the maze in pixels is  $32 \times 6 = 192$  pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 96 pixels in the horizontal and 96 pixels in the vertical, is greater than a third of 128 pixels (64 px),

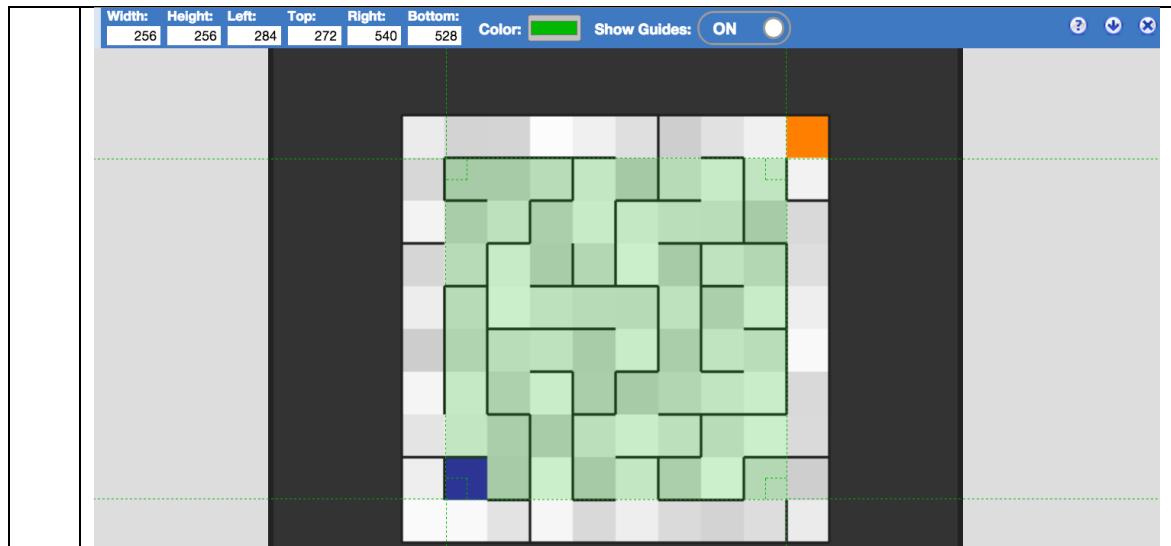
the exit tile and player are greater than a third apart, as intended.



In the fifth image below, the width and height of the maze in pixels is  $32 \times 8 = 256$  pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 194 pixels in the horizontal and 98 pixels in the vertical, is greater than a third of 128 pixels (85 px), the exit tile and player are greater than a third apart, as intended.



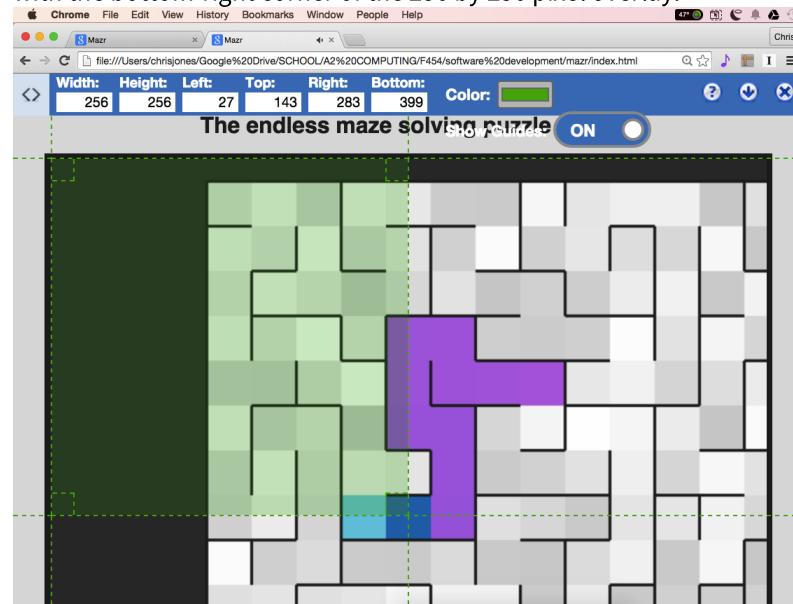
In the sixth image below, the width and height of the maze in pixels is  $32 \times 8 = 256$  pixels horizontally and vertically. As the distance between the player and exit tile's similar corners, 256 pixels in both directions, is greater than a third of 128 pixels (85 px), the exit tile and player are greater than a third apart, as intended.



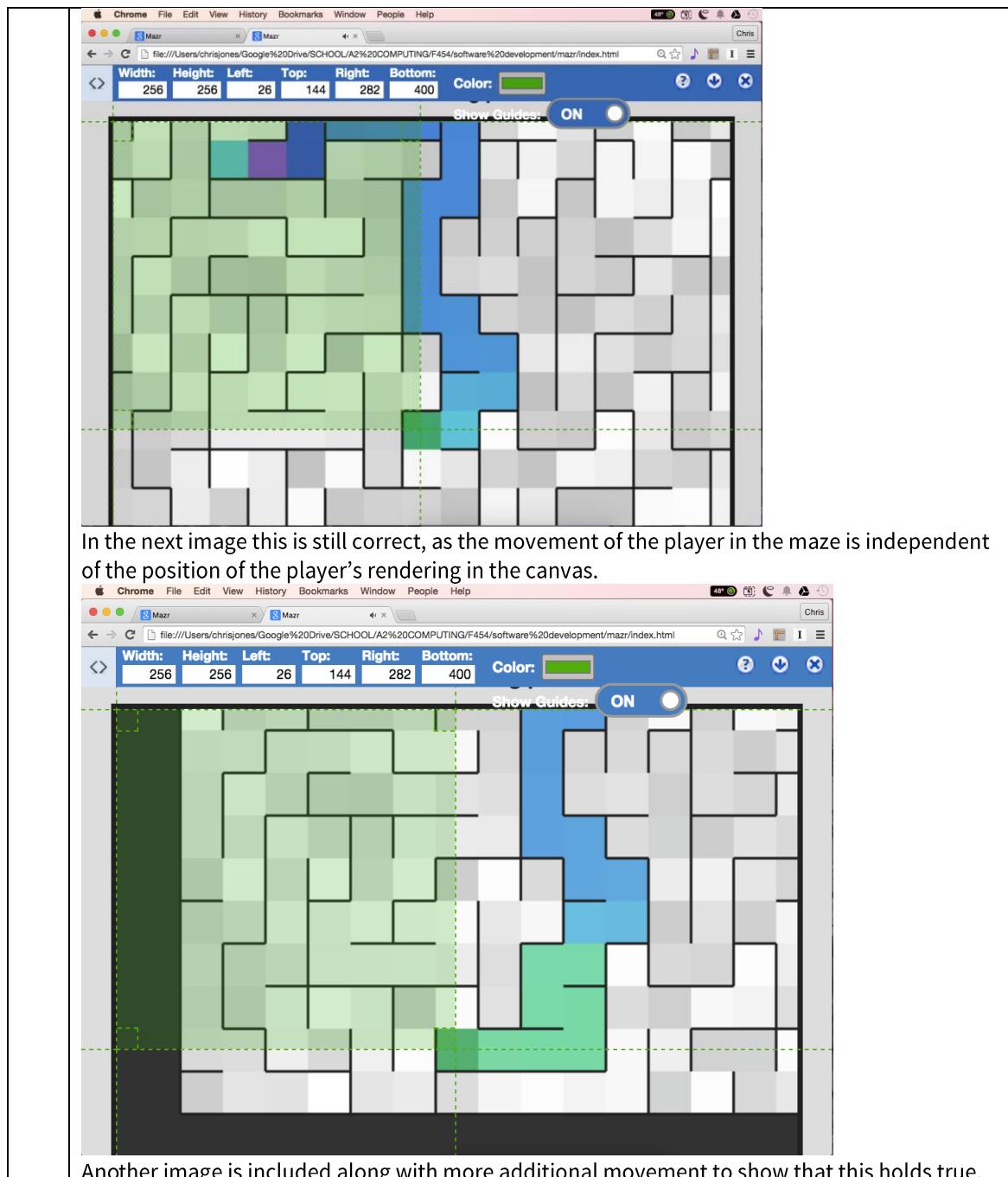
Therefore, as this sample of mazes all have a player and exit tile a third apart, it will be considered that this is a successful test.

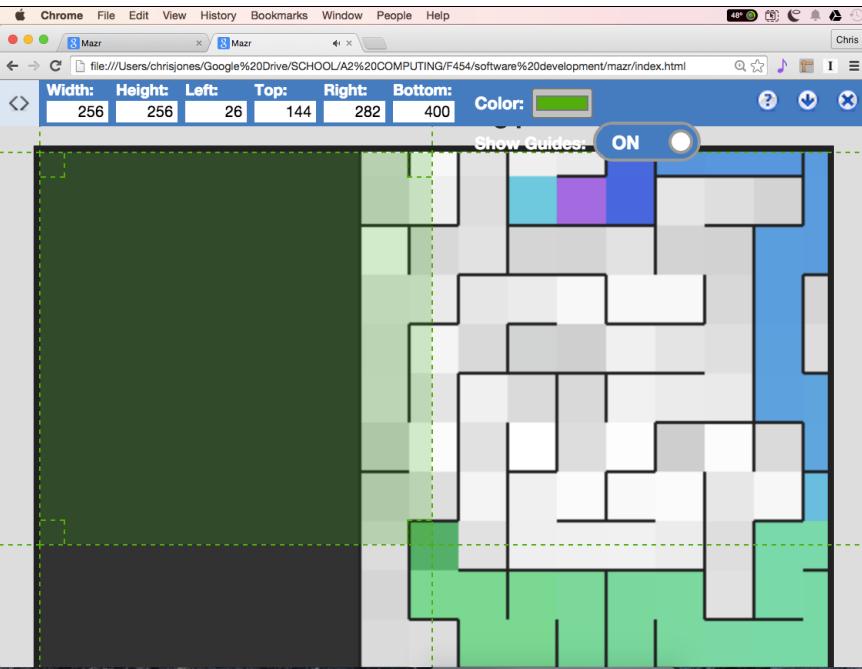
- 34 This test examines a maze, which is larger than the 512 by 512 pixel canvas. The following images use a pixel ruler browser extension that renders a green rectangle on the screen. The top left corner of the 256 by 256 pixel rectangle is placed in the top left corner of the canvas and the bottom right corner therefore resides at the centre of the canvas as the pixel coordinate of the centre is (256, 256). Therefore should the player be centred on this corner of the rectangle it is centred in the canvas and on the camera.

The first image shows that this is correct, as by visual inspection the centre of the player aligns with the bottom-right corner of the 256 by 256 pixel overlay.



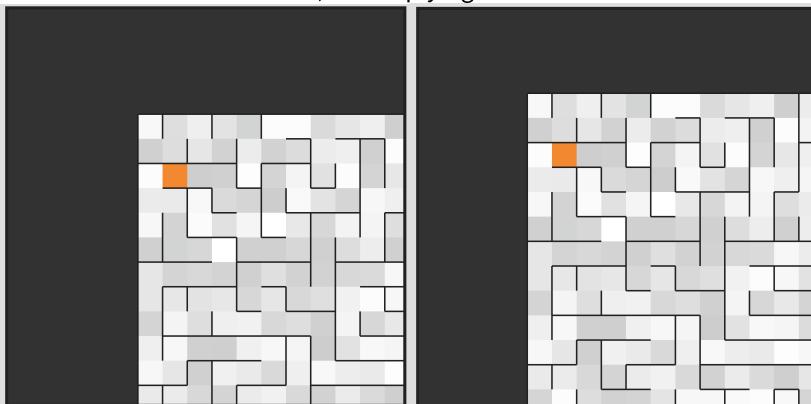
The next image shows that after player movement around the maze the player is still centred on the game window.



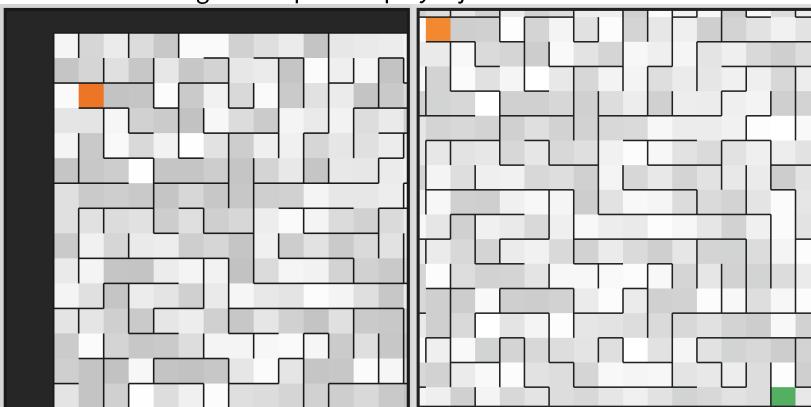


Therefore, for mazes larger than can wholly fit inside the view of the canvas, the player remains centred in the canvas and the camera follows the player around the maze, as intended.

- 35 The following images were taken in time intervals in order to determine whether the camera pans over from the exit tile to the maze when a maze is loaded that is larger than that which can wholly fit inside the canvas. In the first two images below, the exit tile initially starts at the centre of the canvas and then in the first image it can be seen that panning downward right has occurred. This becomes more noticeable in the second image as the exit tile is shown further up and to the left of the canvas (thus implying the camera has moved down and right).

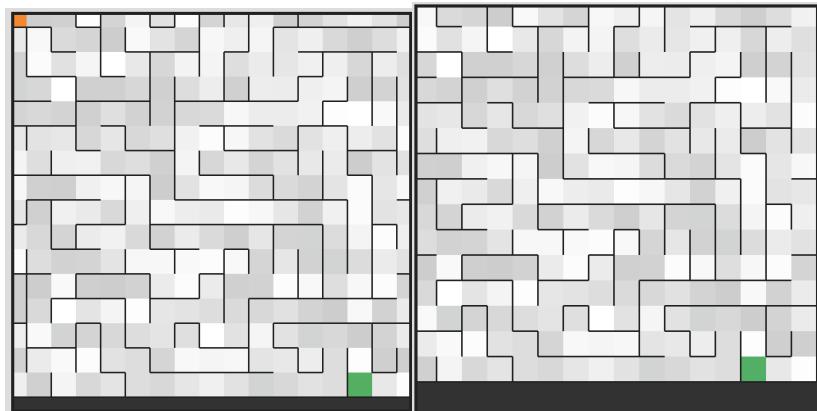


The first of the next two images show the camera panning downwards and to the right and then in the second image the top of the player just becomes visible.

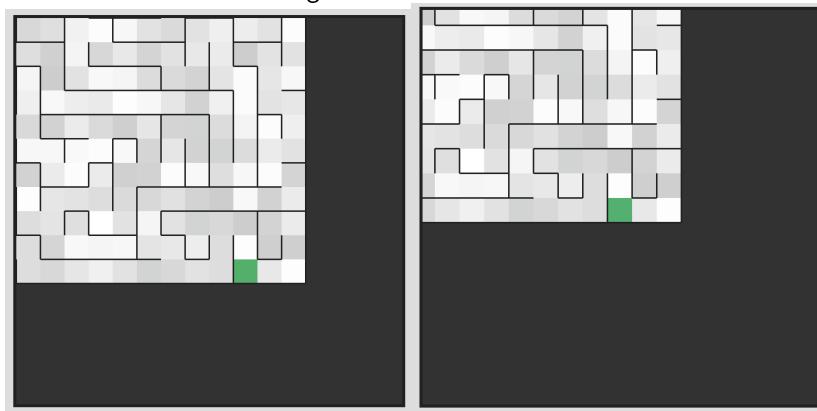


The next two images show the exit tile finally lost from view as the camera pans over from the

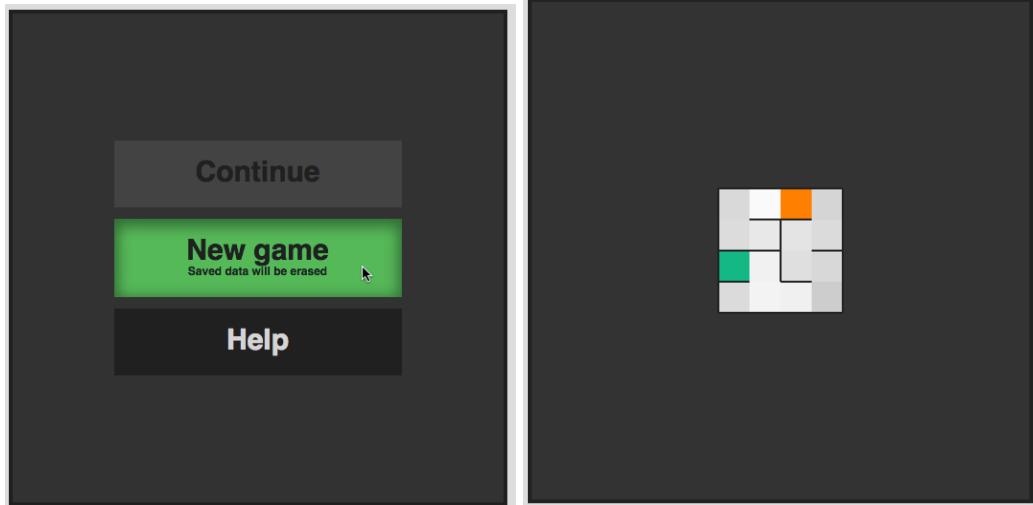
exit tile.



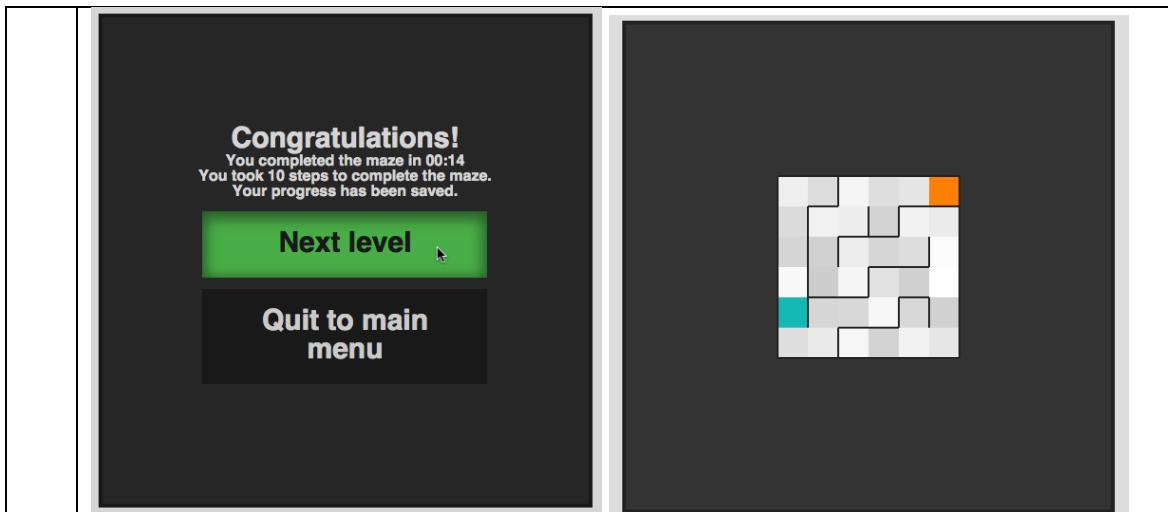
Finally, in the last two images, the canvas has a view of the player, which it eventually centres on as shown in the last image.



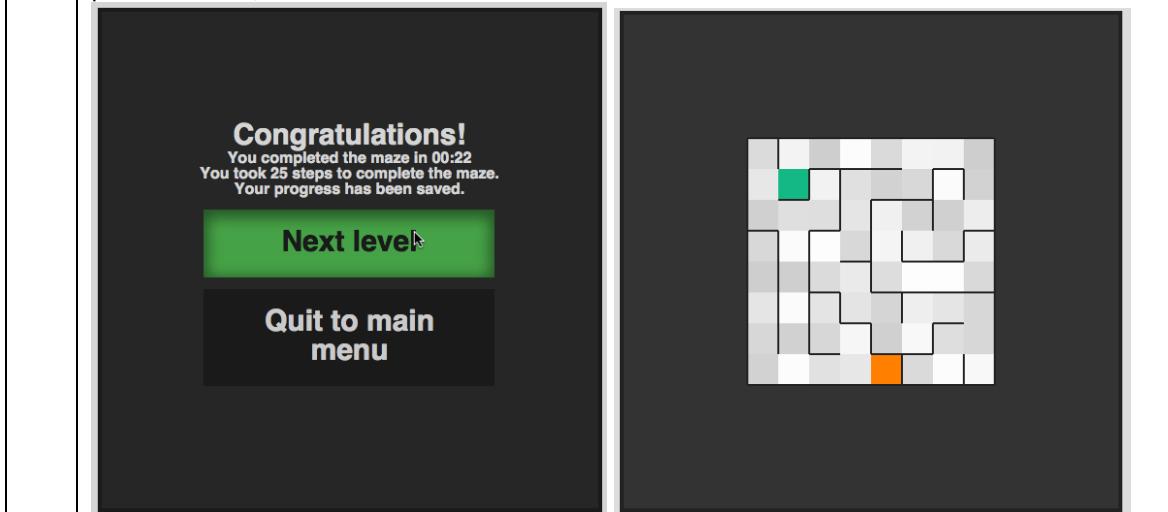
- 36 In the first two images below, the new game button is pressed and the first maze is displayed. This is a 4 by 4 tile maze.



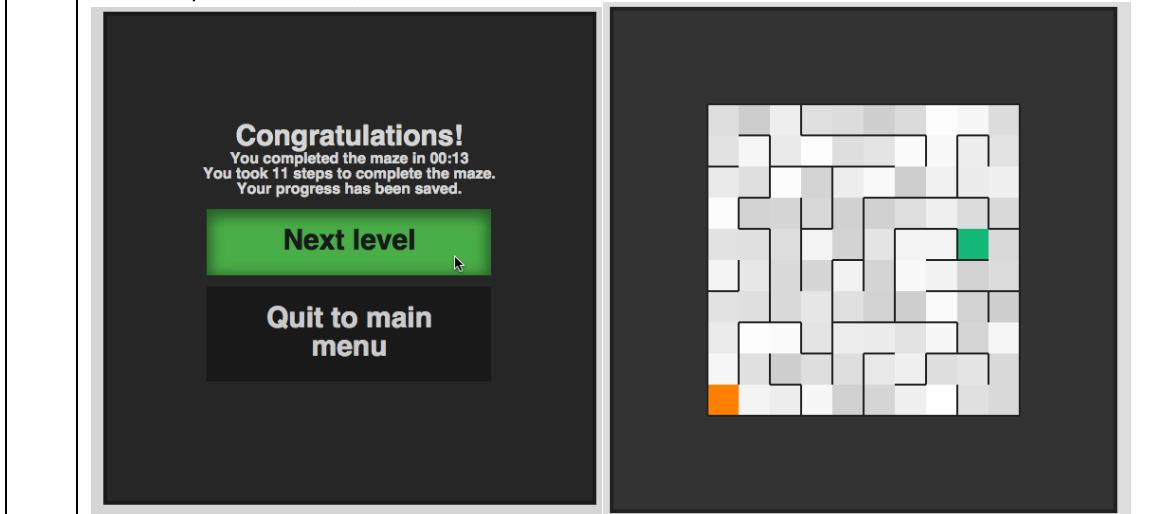
After the first maze is completed, the end of level menu is displayed, as shown in the left image below. The next level button is pressed and the next maze is displayed, as shown in the right image below. This is a 6 by 6 tile maze. Therefore it is 2 tiles larger in each dimension than the previous maze, as intended.



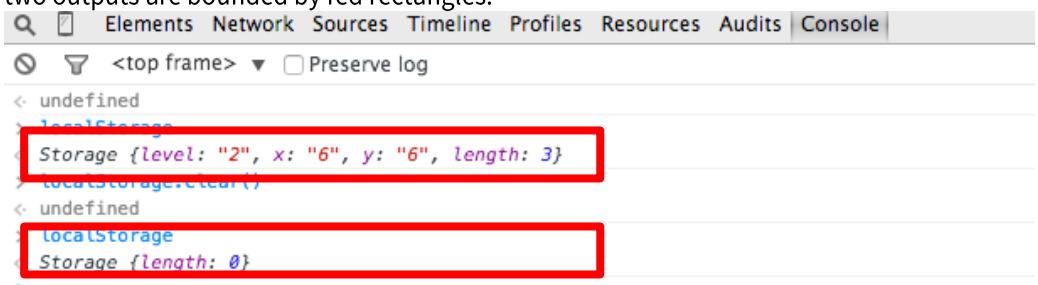
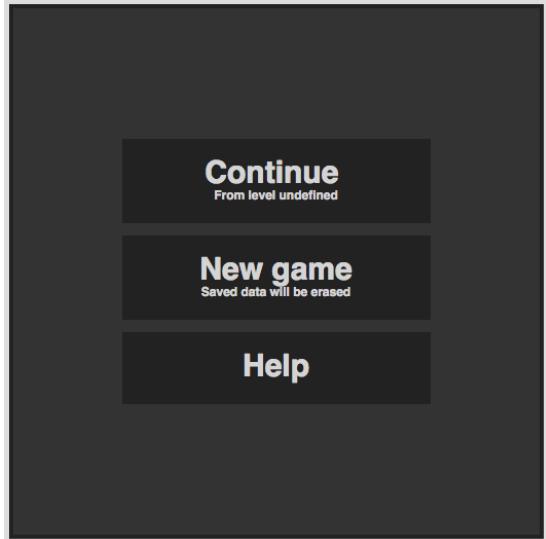
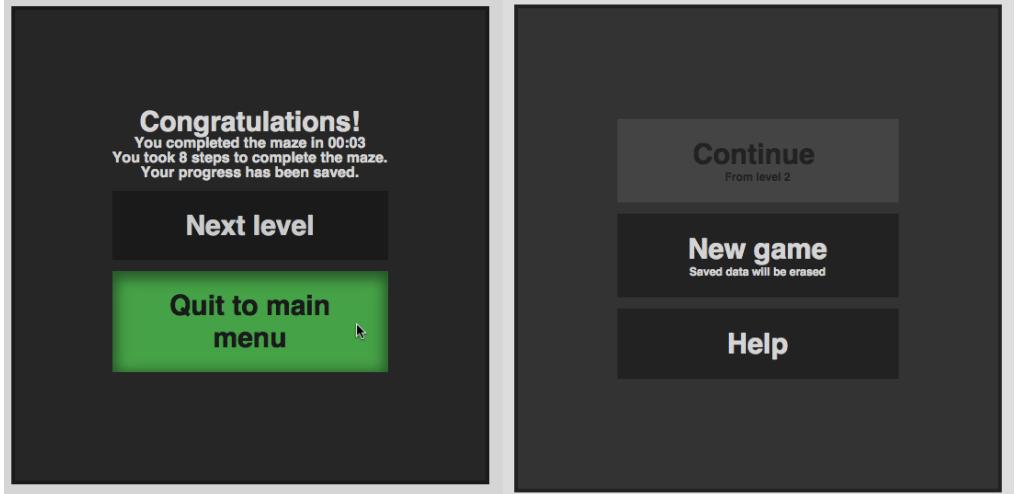
The last maze is completed and the end of level menu displayed, as shown in the left image below. The next level button is pressed and the next maze is displayed, as shown in the right image below. This is an 8 by 8 tile maze. Therefore it is 2 tiles larger in each dimension than the previous maze, as intended.

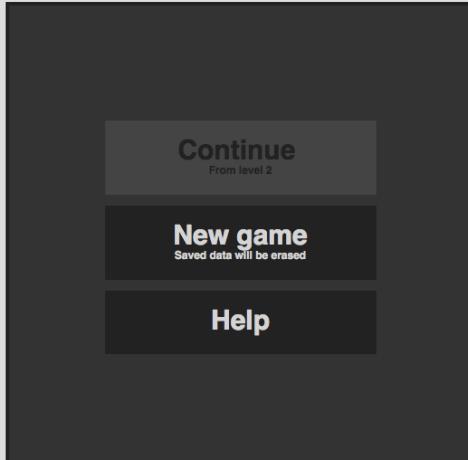
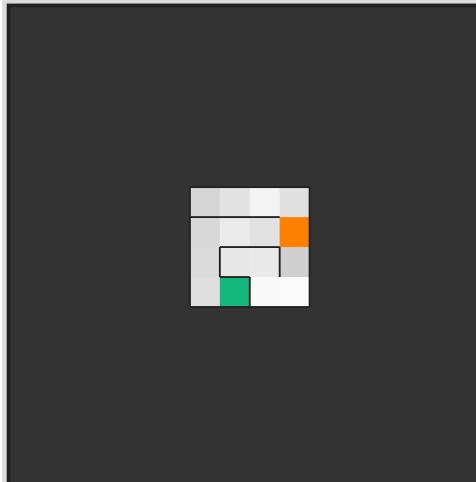
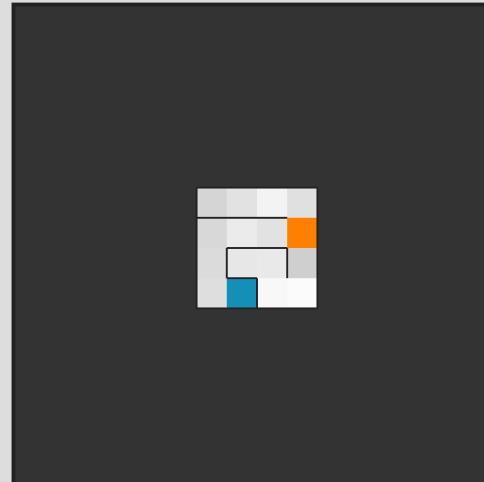
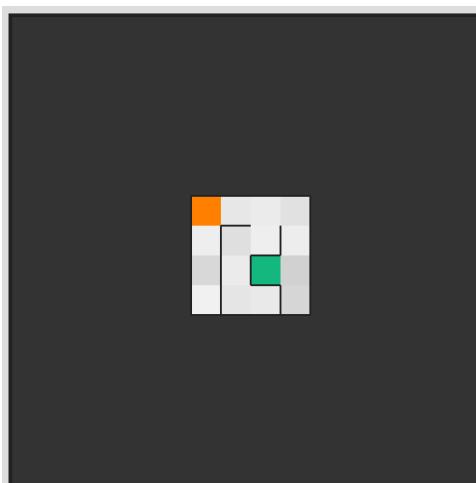
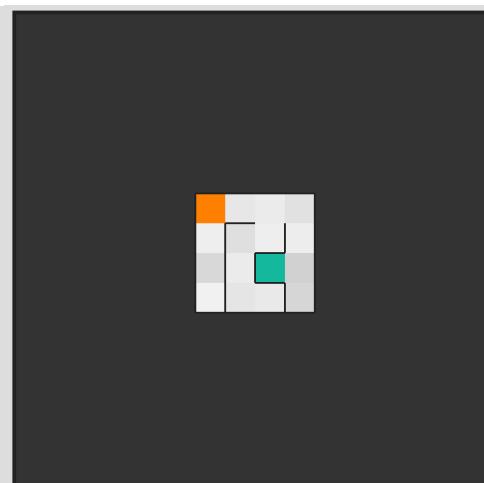


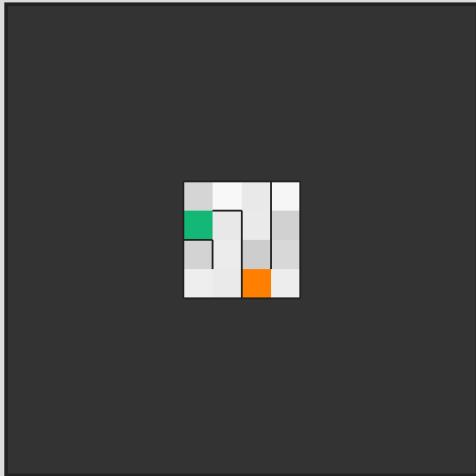
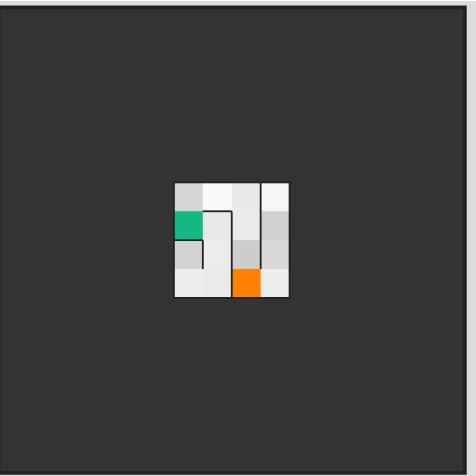
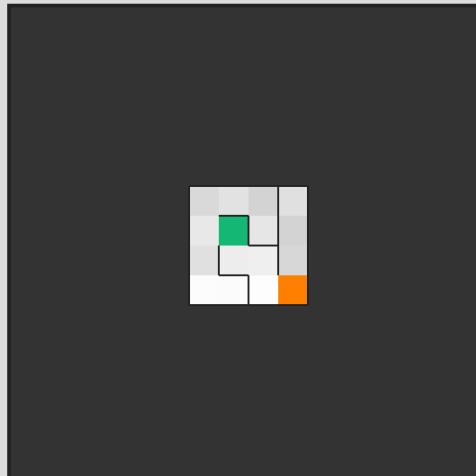
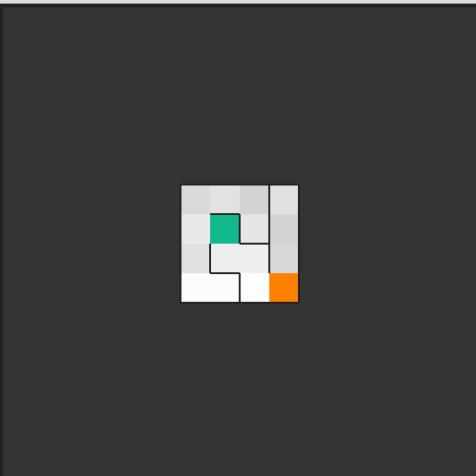
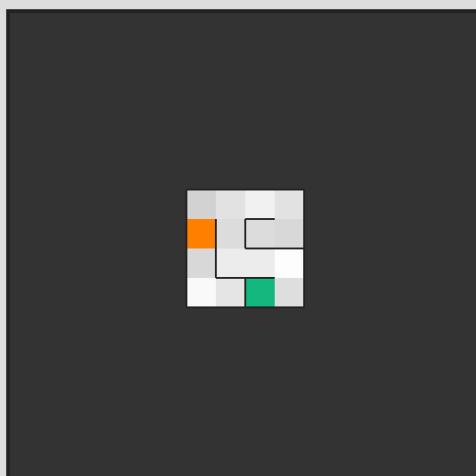
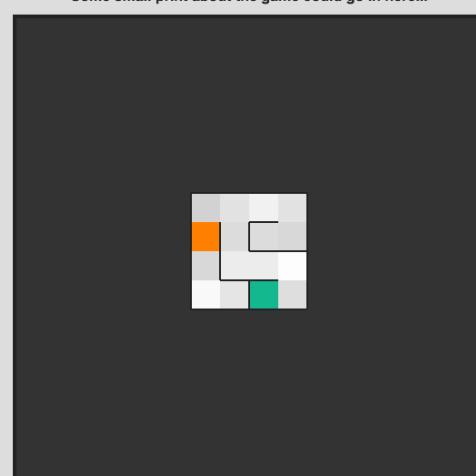
The previous maze is completed and the end of level menu is displayed, as shown in the left image below. The next level button is pressed and the next maze is displayed, as shown in the image to the right. This is a 10 by 10 tile maze. Therefore it is 2 tiles larger in both dimensions than the previous maze, as intended.

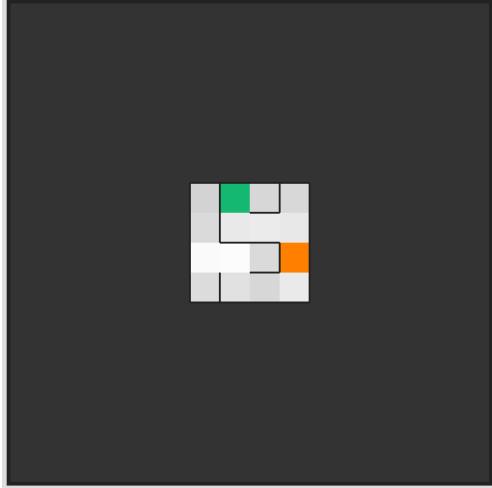
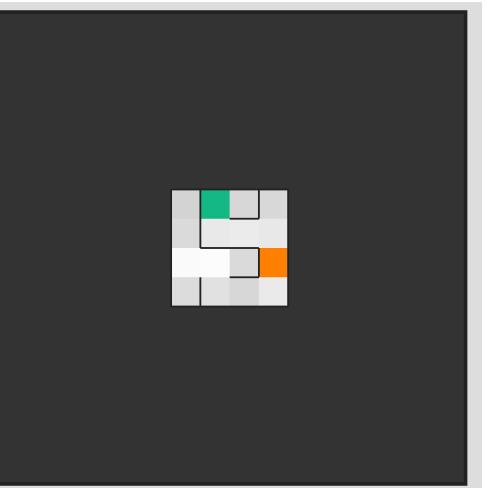
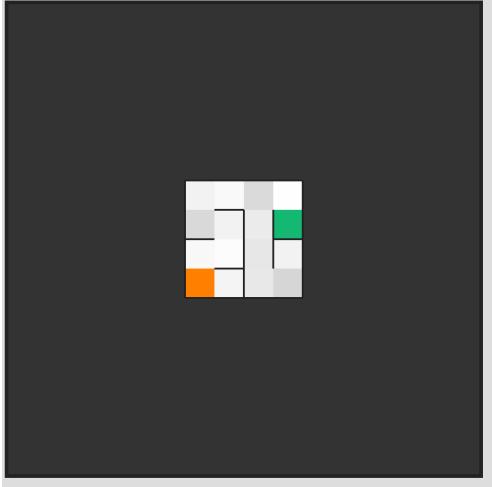
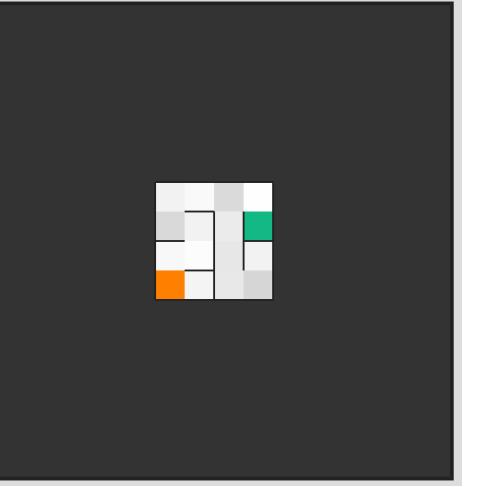
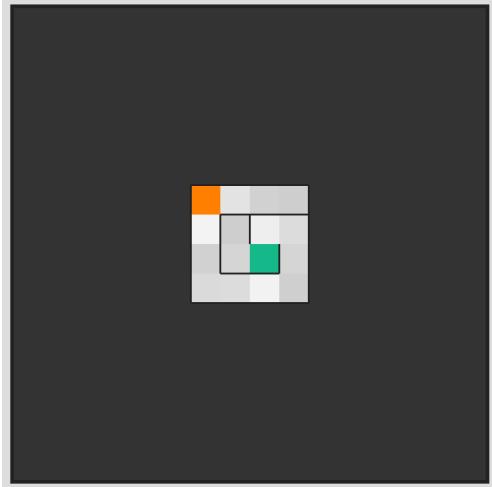
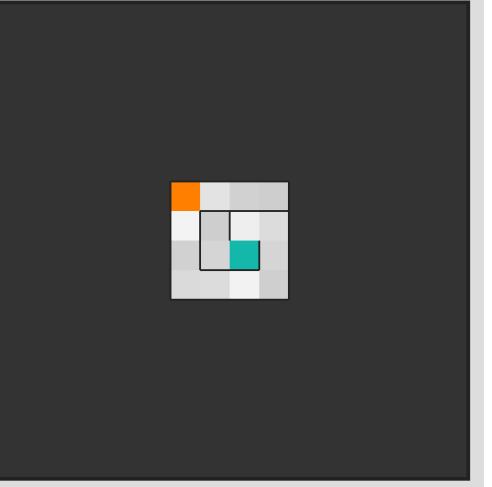


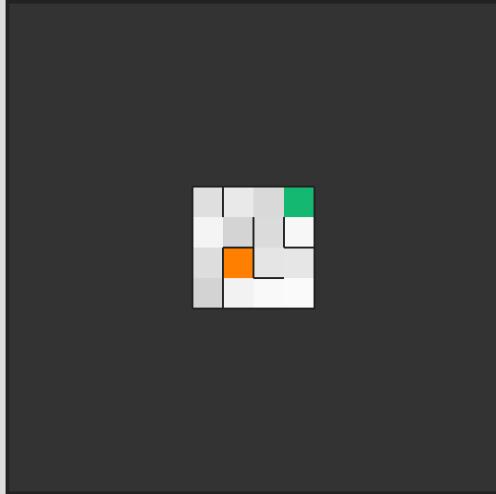
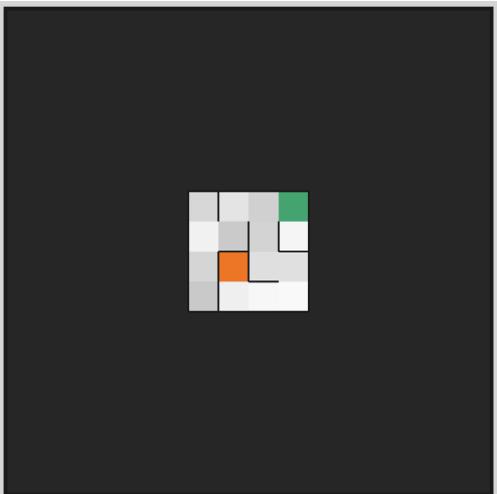
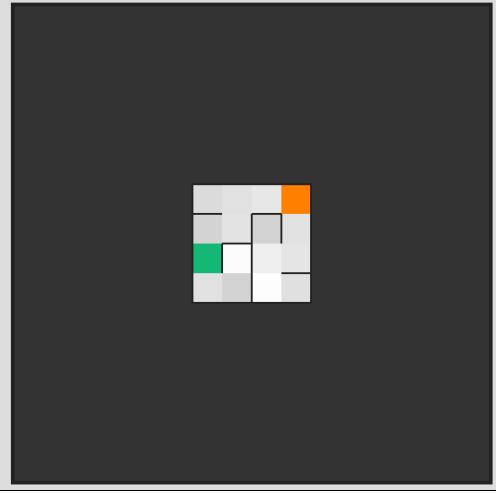
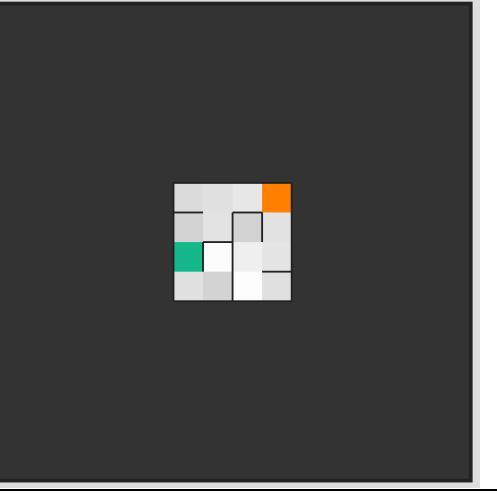
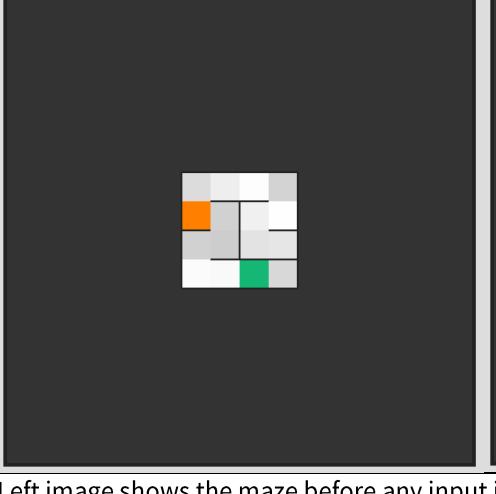
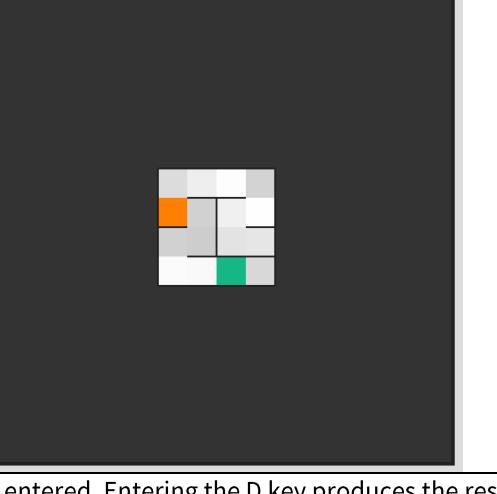
37 First image below shows a screenshot of the JavaScript console in the web browser. Note that

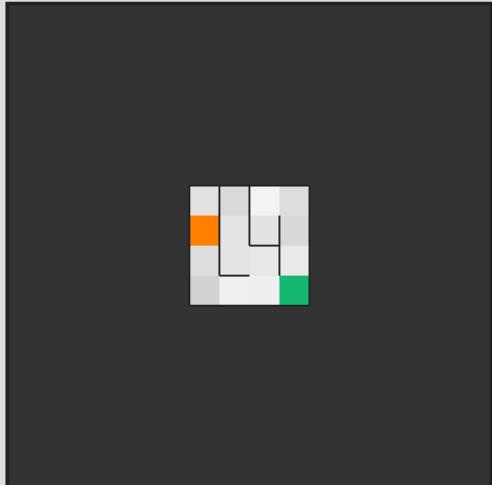
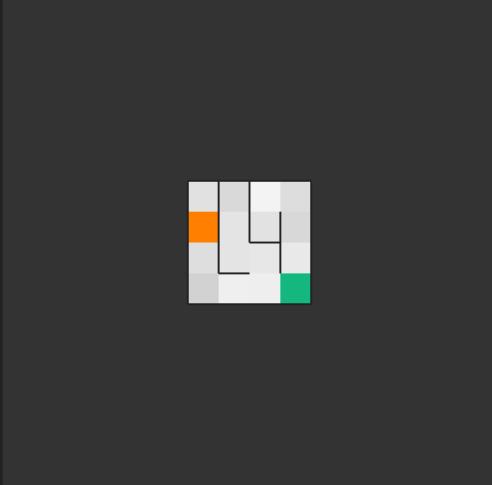
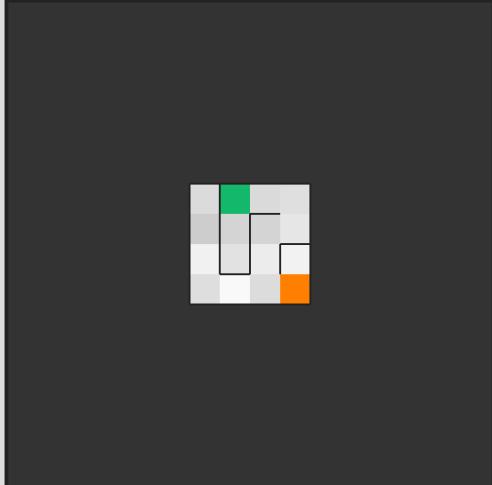
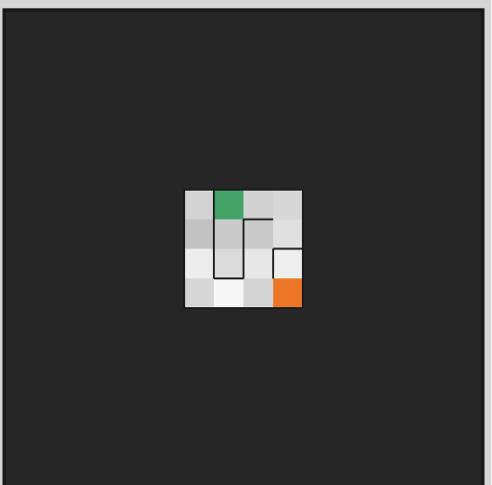
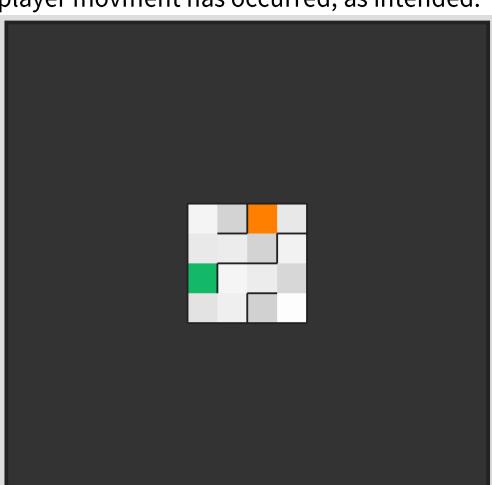
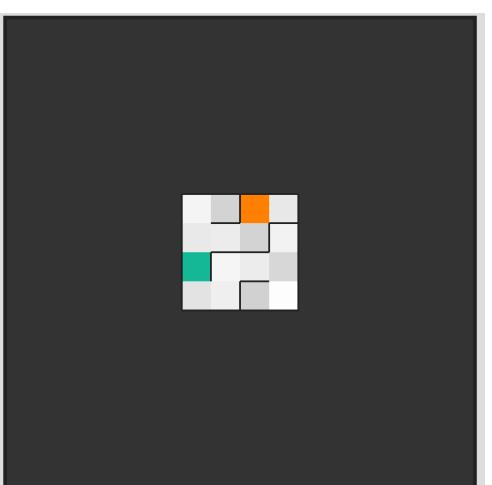
	<p>command shows there is Local Storage present and then the Local Storage is deleted. These two outputs are bounded by red rectangles.</p>  <pre> Elements Network Sources Timeline Profiles Resources Audits Console localStorage &gt; localStorage &lt; undefined localStorage &gt; localStorage.clear() &lt; undefined localStorage &gt; localStorage &lt; undefined localStorage &gt; localStorage &lt; undefined localStorage &gt; </pre>
	<p>The image below shows that, after a browser refresh, the continue button displays the text “From level undefined” and is still enabled. This therefore is an unintended bug and thus results in a failure.</p> 
38	<p>The image on the left shows the Quit to main menu button on the end of level menu being pressed. The image on the right shows the main menu that is displayed once the button has been pressed. However, the continue button is disabled which is unintended behaviour. Therefore this test has failed.</p> 
39	<p>The image on the left shows the Quit to main menu button being pressed on the pause menu. The main menu is then shown, as seen in the image on the right. However, the continue button is disabled which is undesirable behaviour. Therefore this test is unsuccessful.</p>

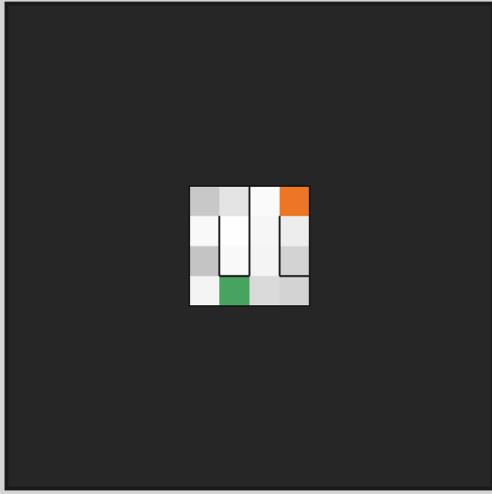
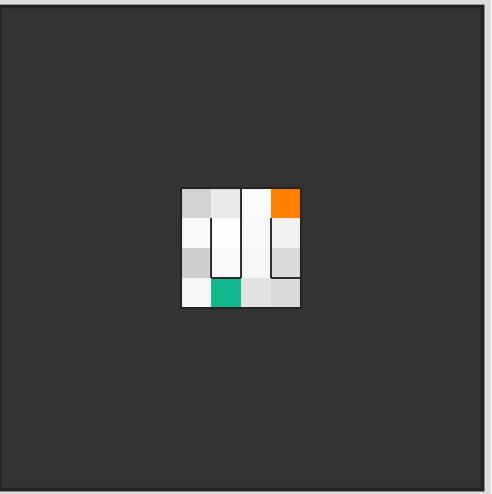
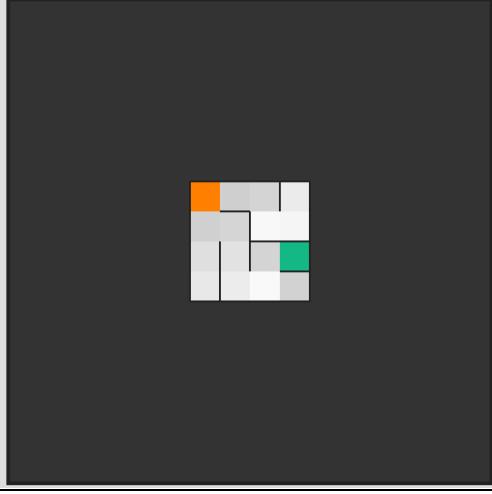
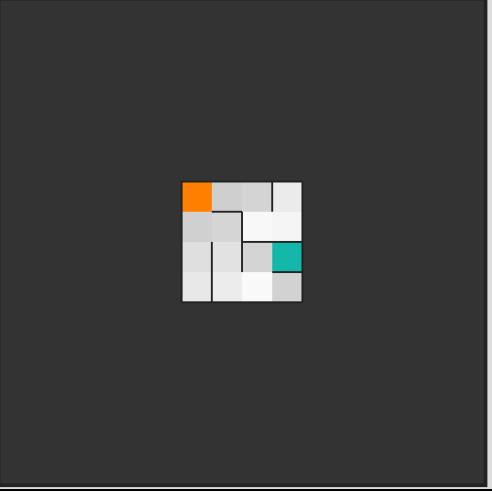
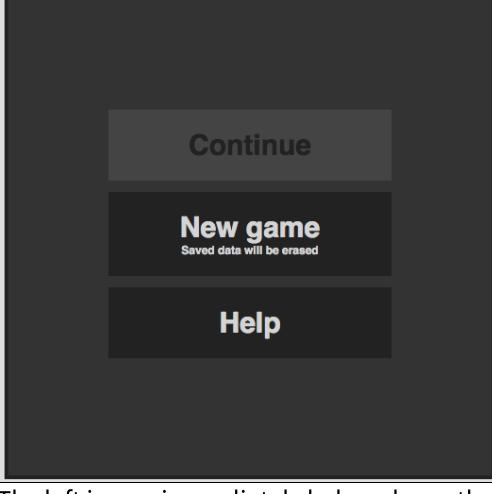
			
40	Left image shows the maze before any input is entered. Upon entering the W key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.		
41	Left image shows the maze before any input is entered. Upon entering the A key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.		
42	Left image shows the maze before any input is entered. Upon entering the S key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.		

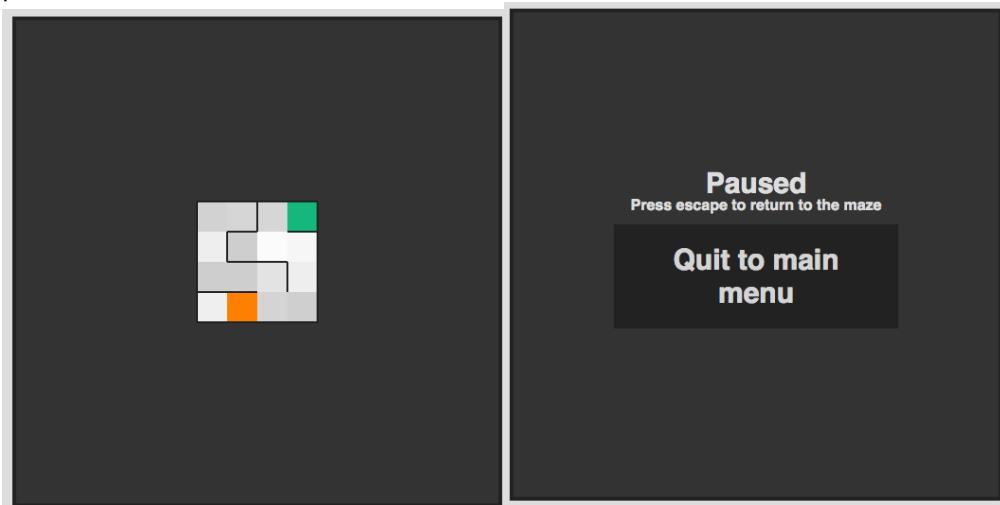
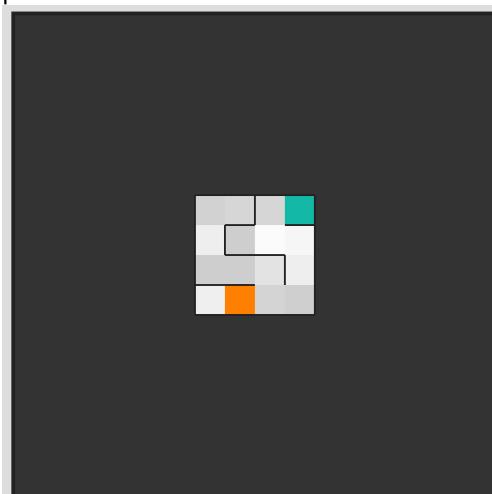
	 A 4x4 grid representing a maze. The top-left cell contains a green player character. The path consists of white cells, with grey cells representing walls and orange cells representing obstacles. The player is positioned at the top-left corner.	 The same 4x4 grid as the left image. The player character has moved one cell to the right, but is now blocked by an orange wall, so no further movement occurs.	
43	Left image shows the maze before any input is entered. Upon entering the D key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.	 A 4x4 grid representing a maze. The top-left cell contains a green player character. The path consists of white cells, with grey cells representing walls and orange cells representing obstacles. The player is positioned at the top-left corner.	 The same 4x4 grid as the left image. The player character has moved one cell to the right, but is now blocked by an orange wall, so no further movement occurs.
44	Left image shows the maze before any input is entered. Upon entering the Up arrow key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.	 A 4x4 grid representing a maze. The top-left cell contains a green player character. The path consists of white cells, with grey cells representing walls and orange cells representing obstacles. The player is positioned at the top-left corner.	 The same 4x4 grid as the left image. The player character has moved one cell upwards, but is now blocked by an orange wall, so no further movement occurs.
45	Left image shows the maze before any input is entered. Upon entering the Left arrow key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.		

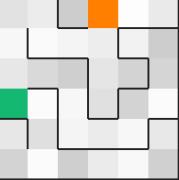
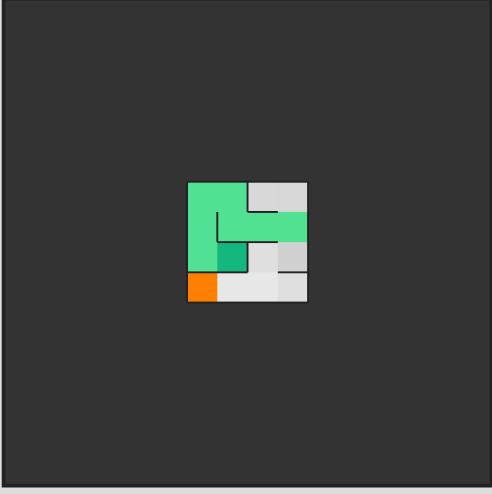
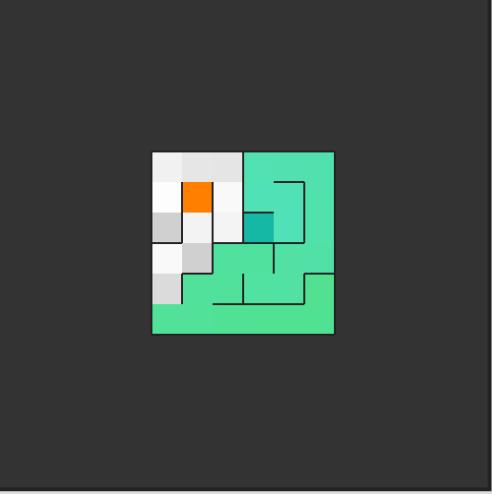
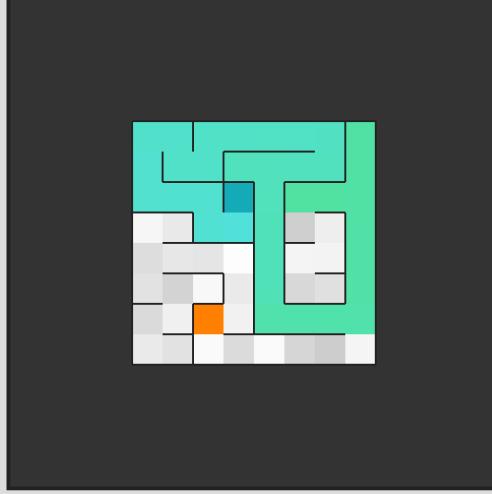
	 A 5x5 grid representing a maze. The player character (a green square) is at position (3, 2). There are orange squares at (1, 4), (2, 4), and (4, 4), which are walls. A grey square at (3, 1) is a floor tile.  The same 5x5 grid. The player character has moved to position (3, 3). The wall at (4, 4) is now highlighted in red, indicating it is an impassable barrier.
46	Left image shows the maze before any input is entered. Upon entering the Down arrow key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.  A 5x5 grid representing a maze. The player character (a green square) is at position (3, 2). There are orange squares at (1, 4), (2, 4), and (4, 4), which are walls. A grey square at (3, 1) is a floor tile.  The same 5x5 grid. The player character has moved to position (3, 3). The wall at (4, 4) is now highlighted in red, indicating it is an impassable barrier.
47	Left image shows the maze before any input is entered. Upon entering the Right arrow key, as shown in the right image (where the path is blocked by a wall), no player movement has occurred, as intended.  A 5x5 grid representing a maze. The player character (a green square) is at position (3, 2). There are orange squares at (1, 4), (2, 4), and (4, 4), which are walls. A grey square at (3, 1) is a floor tile.  The same 5x5 grid. The player character has moved to position (3, 3). The wall at (4, 4) is now highlighted in red, indicating it is an impassable barrier.
48	Left image shows the maze before any input is entered. Entering the W key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.

	 A 5x5 grid representing a maze. The top row is black. The bottom row has a green square at position (1,5), a white square at (2,5), a grey square at (3,5), an orange square at (4,5), and a white square at (5,5). The middle three rows (2, 3, 4) have grey squares at (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), and (4,4). The center cell (3,3) is white.	 The same 5x5 grid as the left image, but the orange square at (4,5) is now at the edge of the grid at (4,4), indicating it cannot move further right.
49	Left image shows the maze before any input is entered. Entering the A key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.	
	 A 5x5 grid representing a maze. The top row is black. The bottom row has a green square at position (1,5), a white square at (2,5), a grey square at (3,5), an orange square at (4,5), and a white square at (5,5). The middle three rows (2, 3, 4) have grey squares at (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), and (4,4). The center cell (3,3) is white.	 The same 5x5 grid as the left image, but the orange square at (4,5) is now at the edge of the grid at (4,4), indicating it cannot move further right.
50	Left image shows the maze before any input is entered. Entering the S key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.	
	 A 5x5 grid representing a maze. The top row is black. The bottom row has a green square at position (1,5), a white square at (2,5), a grey square at (3,5), an orange square at (4,5), and a white square at (5,5). The middle three rows (2, 3, 4) have grey squares at (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), and (4,4). The center cell (3,3) is white.	 The same 5x5 grid as the left image, but the orange square at (4,5) is now at the edge of the grid at (4,4), indicating it cannot move further right.
51	Left image shows the maze before any input is entered. Entering the D key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.	

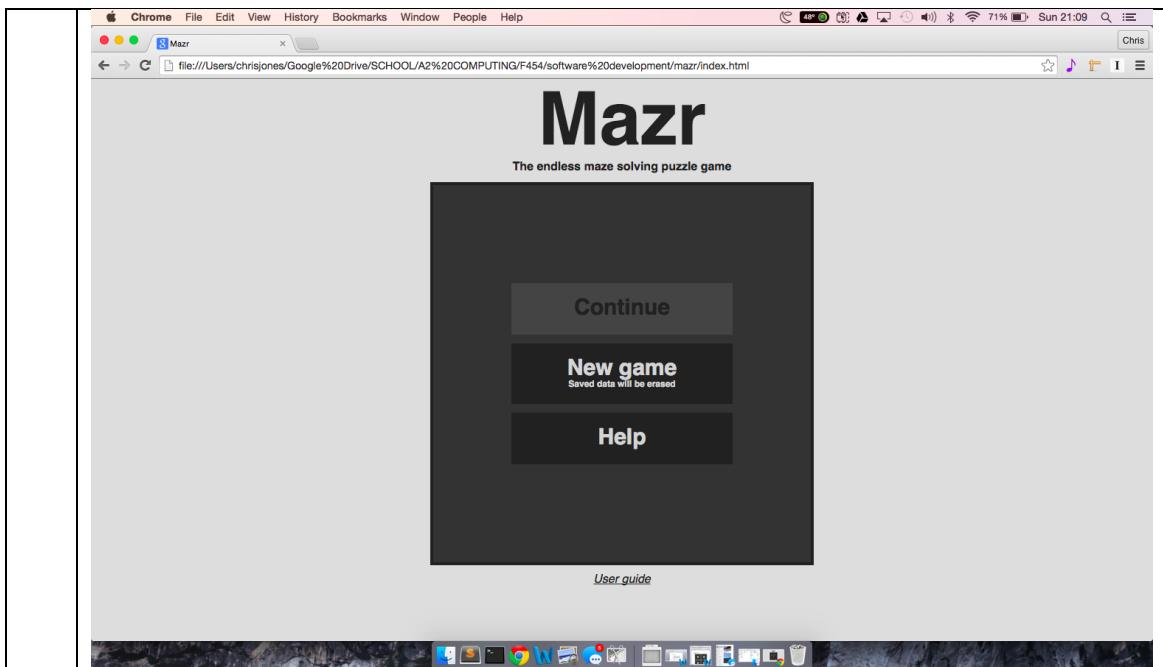
	 A 4x4 grid representing a maze. The top-left cell is orange, indicating the player's current position. The bottom-right cell is green, indicating the goal. The middle row contains two grey cells, and the middle column contains one grey cell, creating a central barrier.	 The same 4x4 grid as the left image. The player's orange cell has moved up to the second row, first column. The path from the player to the goal is now blocked by the edge of the grid, preventing further movement.	
52	Left image shows the maze before any input is entered. Entering the Up arrow key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.		
	 A 4x4 grid representing a maze. The top-left cell is orange, indicating the player's current position. The bottom-right cell is green, indicating the goal. The middle row contains two grey cells, and the middle column contains one grey cell, creating a central barrier.	 The same 4x4 grid as the left image. The player's orange cell has moved left to the second column, first row. The path from the player to the goal is now blocked by the edge of the grid, preventing further movement.	
53	Left image shows the maze before any input is entered. Entering the Left arrow key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.		
	 A 4x4 grid representing a maze. The top-left cell is orange, indicating the player's current position. The bottom-right cell is green, indicating the goal. The middle row contains two grey cells, and the middle column contains one grey cell, creating a central barrier.	 The same 4x4 grid as the left image. The player's orange cell has moved down to the third row, first column. The path from the player to the goal is now blocked by the edge of the grid, preventing further movement.	
54	Left image shows the maze before any input is entered. Entering the Down arrow key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.		

		
55	Left image shows the maze before any input is entered. Entering the Right arrow key produces the result shown in the right image (where the path is blocked by the edge of the maze) and no player movement has occurred, as intended.	
56	The image on the left shows the main menu. The W, A, S, D, Up arrow, Left arrow, Down arrow and Right arrow keys were pressed multiple times when on the main menu. Then, the New game button was pressed. Note that there is no trail left in the maze by the player which would suggest the movement of the player has not been affected by the input to the system on the main menu. This is desirable and intended behaviour.	
57	The left image immediately below shows that a maze is being displayed. Upon pressing the escape key the pause menu is displayed, as shown in the image on the right. The W, A, S, D, Up	

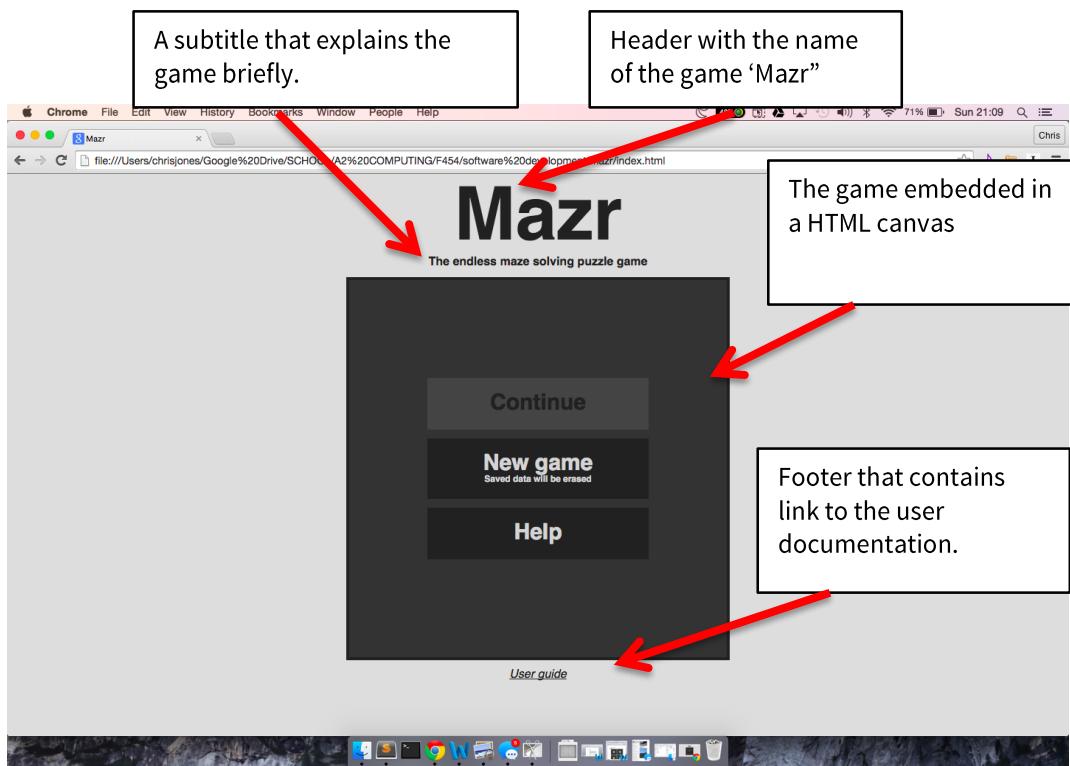
	<p>arrow, Left arrow, Down arrow and Right arrow keys were pressed multiple times when on the pause menu.</p> 
58	<p>Then, the escape key was pressed again to resume the maze, as shown in the left image below. Note that no player movement has occurred after the input was entered to the system on the pause menu. This is desirable and intended behaviour.</p> 

	<p><b>Congratulations!</b> You completed the maze in 00:02 You took 6 steps to complete the maze. Your progress has been saved.</p> <p><b>Next level</b></p> <p><b>Quit to main menu</b></p>	
59	<p>Note the following four images show mazes in which input has been entered in the form of the WASD and arrow keys in order to move the player around the maze. Note in each of the images there is a trail that is left by the movement of the player. This is a colour that changes with respect to the player. As, via visual inspection, it can easily be determined a trail is left by the player that this functions as intended.</p>	
		
		
60	<p>Below are images of a maze taken every few seconds and go left to right and then down sequentially over time. The images show the colour change of the player. Note the player starts as a green tile and then fades through blue and then through to purple. Then the player fades back to blue and back to green again.</p>	

	As the player fades correctly this can be considered as working successfully.		
61	Below is an image that shows the webpage on which the game is hosted visited in a Web Browser. Note from the URL address bar that the file is hosted locally on the system although it is still a HTML document and would be rendered in the same manner were it hosted in a production environment.		



- 62 The previous test's evidence has been replicated here as it shows the contents of the webpage when it is viewed in a browser. The header, subtitle and footer elements are highlighted with labels below.



- 63 The image below shows a selection in the settings of a web browser. Note that JavaScript has been disabled for all websites.

	<p><b>JavaScript</b></p> <p><input type="radio"/> Allow all sites to run JavaScript (recommended)</p> <p><input checked="" type="radio"/> Do not allow any site to run JavaScript</p> <p><a href="#">Manage exceptions...</a></p> <p>With JavaScript now disabled, the webpage on which the game is located was opened in a web browser, as shown in the image below. Note that an error message is displayed as intended and ALL of the menus are rendered in the webpage. This is undesirable behaviour that results in an unsuccessful test.</p>
64	<p>The following images show the element inspector tab in the developer tools of Google Chrome. Each image represents the HTML when a current menu is being viewed. The display attributes are highlighted with a red rectangle in each image to show that the only one menu is being displayed at a given time with the attribute 'inline-flex'. All of the other menus at any give time should have display set to 'none'.</p> <p>Main menu:</p> <pre>&lt;section style="display: block;"&gt;   &lt;div class="canvas-wrapper"&gt;     &lt;canvas id="canvas" class="page" width="512" height="512" style="display: none;"&gt;...&lt;/canvas&gt;     &lt;div class="main-menu page" style="display: inline-flex;"&gt;...&lt;/div&gt;     &lt;div class="help-1 page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="help-2 page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="help-3 page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="pause-menu page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="end-of-level-menu page" style="display: none;"&gt;...&lt;/div&gt;   &lt;/div&gt; &lt;/section&gt;</pre> <p>First help menu:</p> <pre>&lt;section style="display: block;"&gt;   &lt;div class="canvas-wrapper"&gt;     &lt;canvas id="canvas" class="page" width="512" height="512" style="display: none;"&gt;...&lt;/canvas&gt;     &lt;div class="main-menu page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="help-1 page" style="display: inline-flex;"&gt;...&lt;/div&gt;     &lt;div class="help-2 page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="help-3 page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="pause-menu page" style="display: none;"&gt;...&lt;/div&gt;     &lt;div class="end-of-level-menu page" style="display: none;"&gt;...&lt;/div&gt;   &lt;/div&gt; &lt;/section&gt;</pre>

## Second help menu:

```

▼ <div class="canvas-wrapper">
  ► <canvas id="canvas" class="page" width="512" height="512" style="display: none;">...</canvas>
  ► <div class="main-menu page" style="display: none;">...</div>
  ► <div class="help-1 page" style="display: none;">...</div>
  ► <div class="help-2 page" style="display: inline-flex;">...</div>
  ▶ <div class="help-3 page" style="display: none;">...</div>
  ▶ <div class="pause-menu page" style="display: none;">...</div>
  ▶ <div class="end-of-level-menu page" style="display: none;">...</div>
</div>
</section>

```

## Third help menu:

```

▼ <section style="display: block;">
  ▼ <div class="canvas-wrapper">
    ► <canvas id="canvas" class="page" width="512" height="512" style="display: none;">...</canvas>
    ► <div class="main-menu page" style="display: none;">...</div>
    ► <div class="help-1 page" style="display: none;">...</div>
    ► <div class="help-2 page" style="display: none;">...</div>
    ▶ <div class="help-3 page" style="display: inline-flex;">...</div>
    ▶ <div class="pause-menu page" style="display: none;">...</div>
    ▶ <div class="end-of-level-menu page" style="display: none;">...</div>
  </div>
</section>

```

## Pause menu:

```

▼ <div class="canvas-wrapper">
  ► <canvas id="canvas" class="page" width="512" height="512" style="display: none;">...</canvas>
  ► <div class="main-menu page" style="display: none;">...</div>
  ► <div class="help-1 page" style="display: none;">...</div>
  ► <div class="help-2 page" style="display: none;">...</div>
  ▶ <div class="help-3 page" style="display: none;">...</div>
  ▶ <div class="pause-menu page" style="display: inline-flex;">...</div>
  ▶ <div class="end-of-level-menu page" style="display: none;">...</div>
</div>

```

## End of level menu:

```

▼ <section style="display: block;">
  ▼ <div class="canvas-wrapper">
    ► <canvas id="canvas" class="page" width="512" height="512" style="display: none;">...</canvas>
    ► <div class="main-menu page" style="display: none;">...</div>
    ► <div class="help-1 page" style="display: none;">...</div>
    ► <div class="help-2 page" style="display: none;">...</div>
    ► <div class="help-3 page" style="display: none;">...</div>
    ▶ <div class="pause-menu page" style="display: none;">...</div>
    ▶ <div class="end-of-level-menu page" style="display: inline-flex;">...</div>
  </div>
</section>

```

## The canvas (included for completion):

```

▼ <section style="display: block;">
  ▼ <div class="canvas-wrapper">
    ► <div class="main-menu page" style="display: none;">...</div>
    ▶ <div class="help-1 page" style="display: none;">...</div>
    ▶ <div class="help-2 page" style="display: none;">...</div>
    ▶ <div class="help-3 page" style="display: none;">...</div>
    ▶ <div class="pause-menu page" style="display: none;">...</div>
    ▶ <div class="end-of-level-menu page" style="display: none;">...</div>
  </div>
</section>

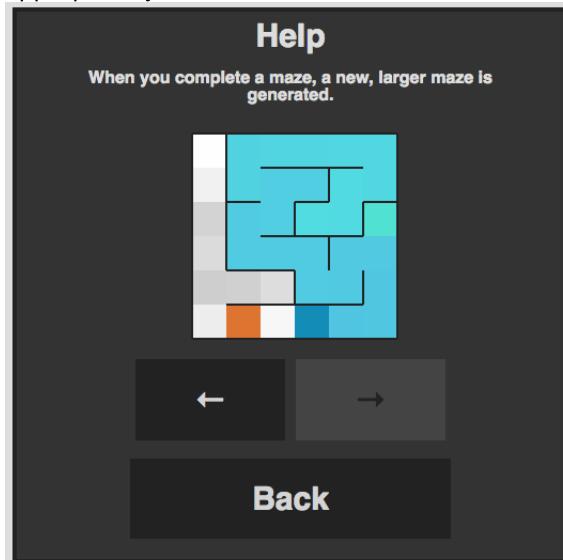
```

Therefore, as all of the menus are present as HTML div elements and only one menu displays at a time, this can be classed as intended behaviour.

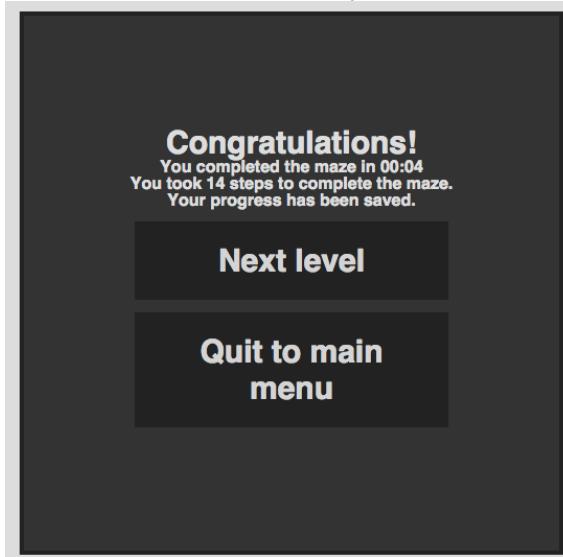
- |    |   |
|----|---|
| 65 | Images below show the main menu (with no saved data present on the left and saved data being present on the right). Note via visual inspection the presence of a continue button (disabled when there is no saved data), a new game button and a help button. |
|----|---|

66	<p>Below is an image of the first help menu. Note by visual inspection that it contains onscreen support along with an associated image, two arrow buttons with the left disabled appropriately and a back button. This is as intended.</p>	
67	<p>Below is an image of the second help menu. Note by visual inspection that it contains onscreen support along with an associated image, two arrow buttons with the neither disabled and a back button. This is as intended.</p>	

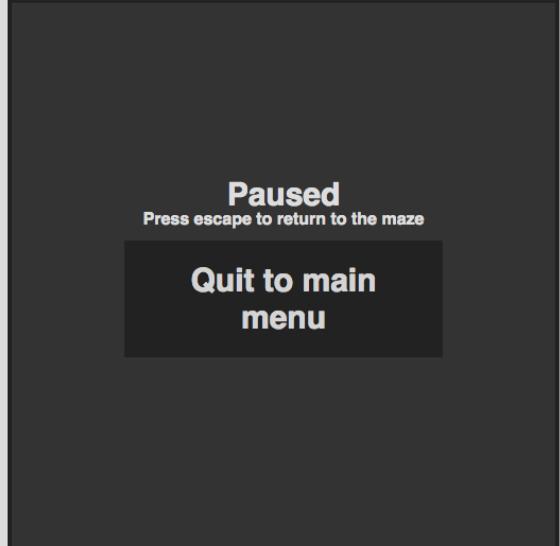
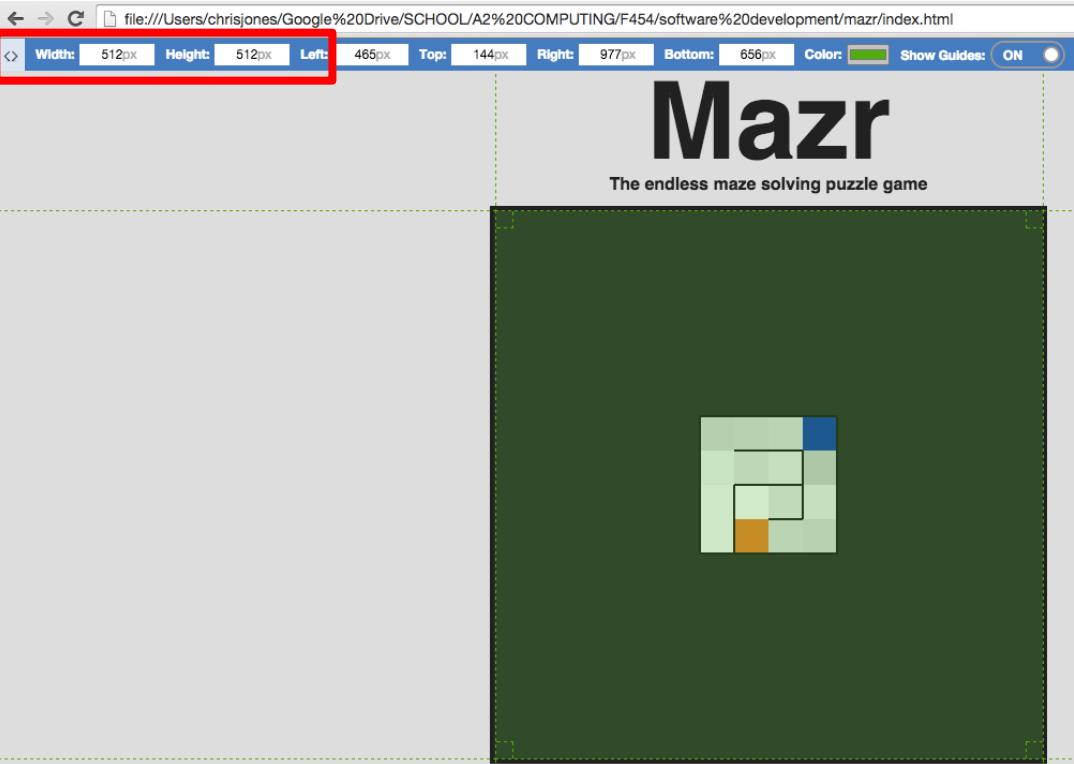
- 68 Below is an image of the third help menu. Note by visual inspection that it contains onscreen support along with an associated image, two arrow buttons with the right disabled appropriately and a back button. This is as intended.

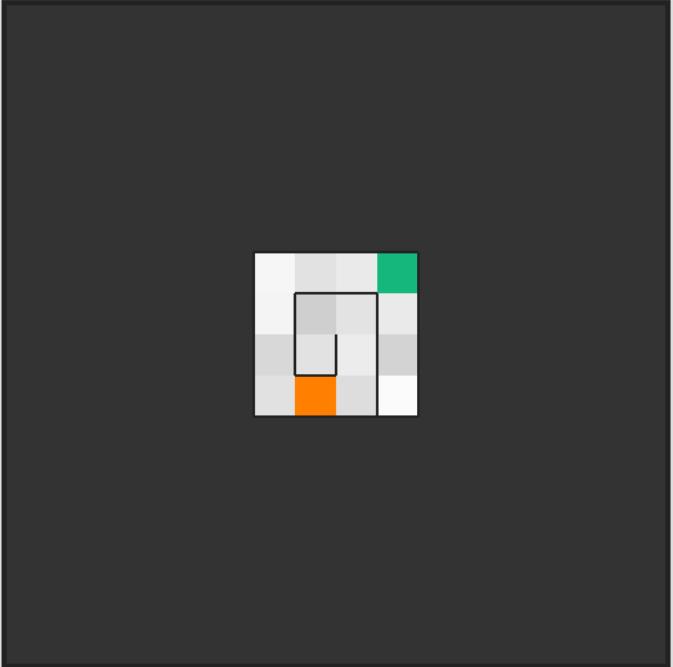
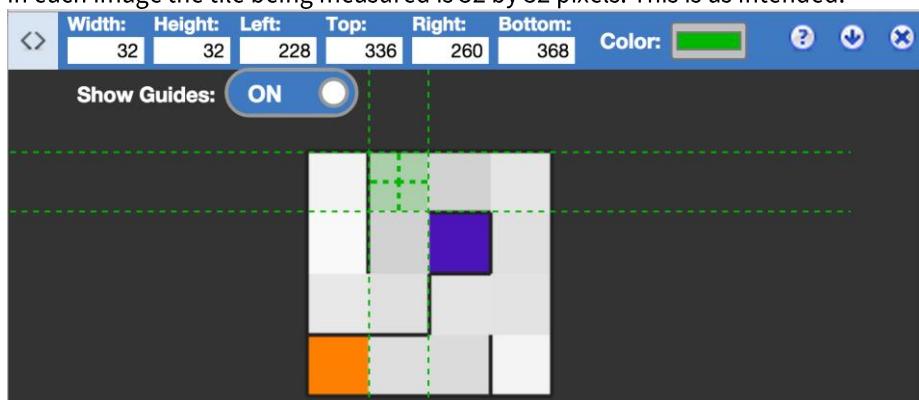


- 69 Below is an image of the end of level menu. Note the presence of text congratulating the user, text displaying the time taken to complete the maze, text displaying the number of steps required to complete a maze and a button that allows the user to go to the next maze and a button that allows the user to quit to the main menu. This is as intended.

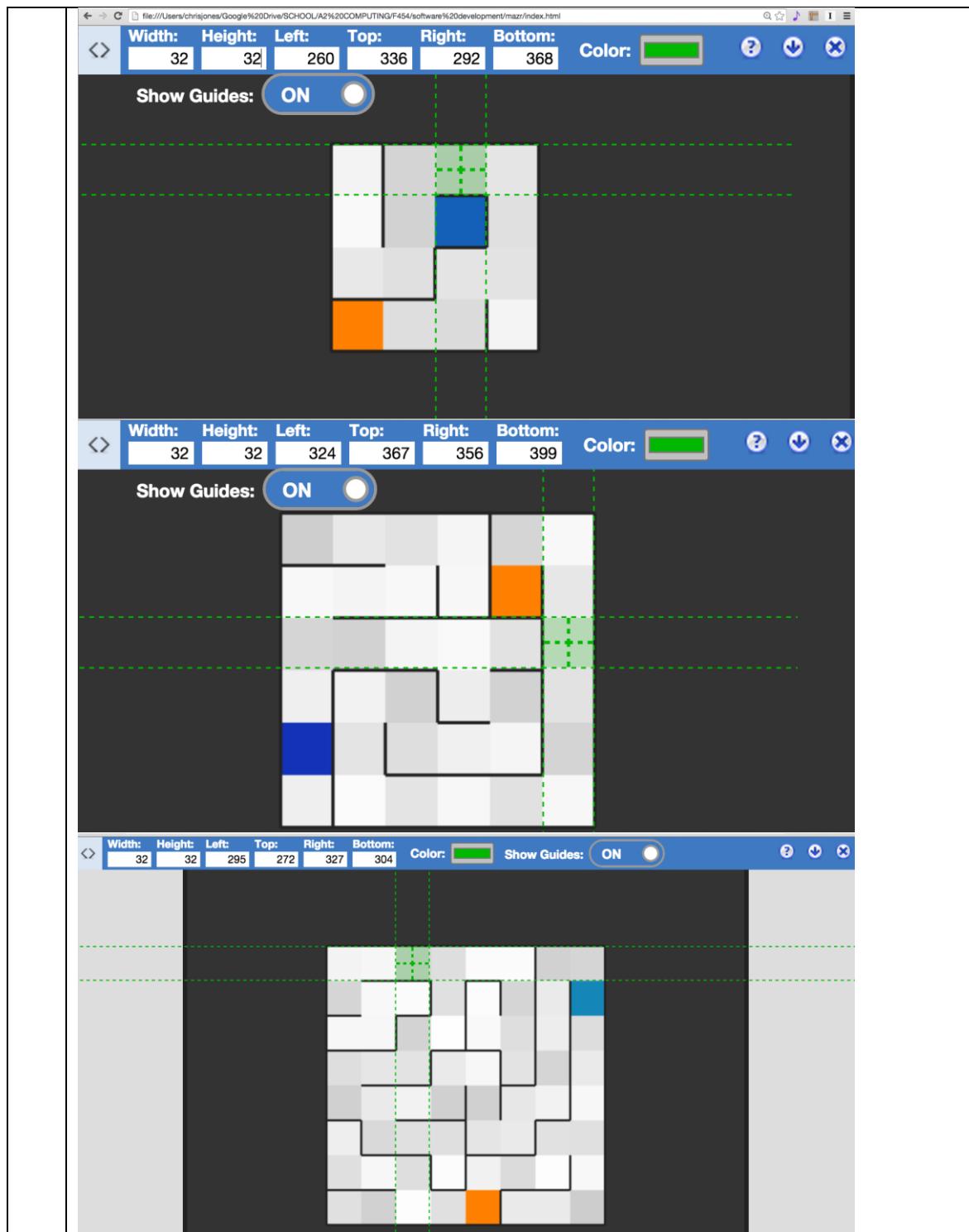


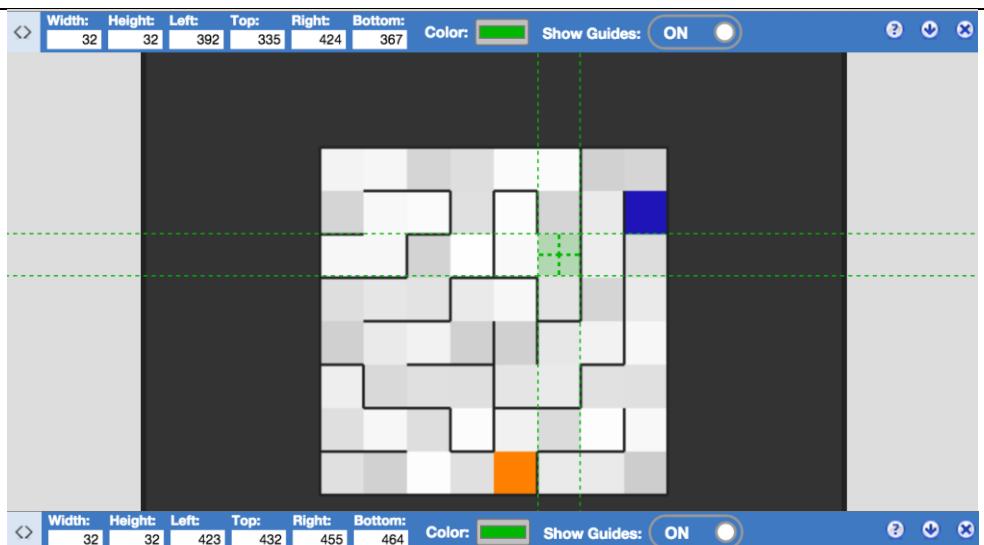
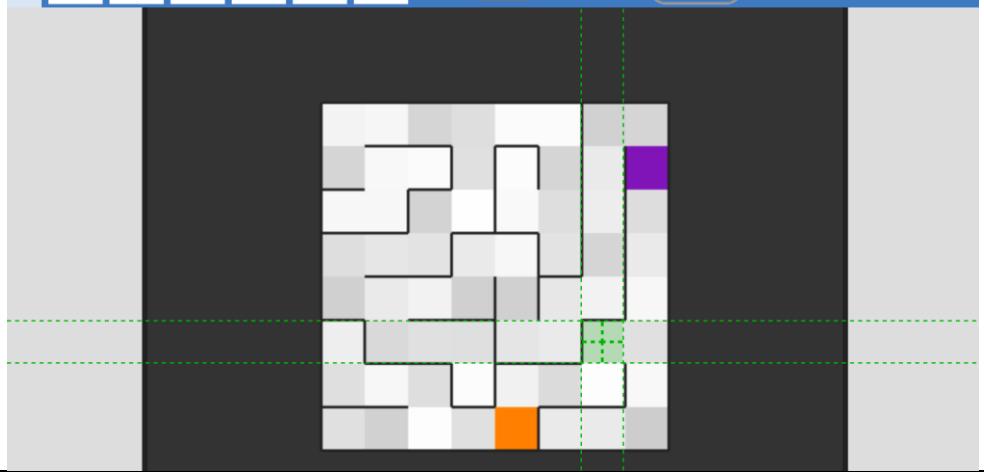
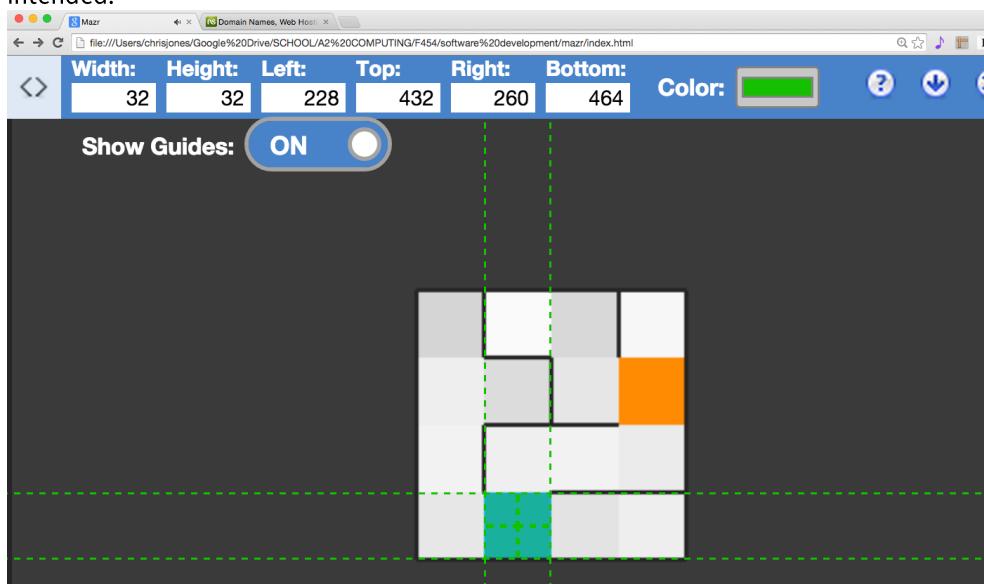
- 70 Below is an image of the pause menu, which contains a button that allows the user to quit to the main menu. This is as intended.

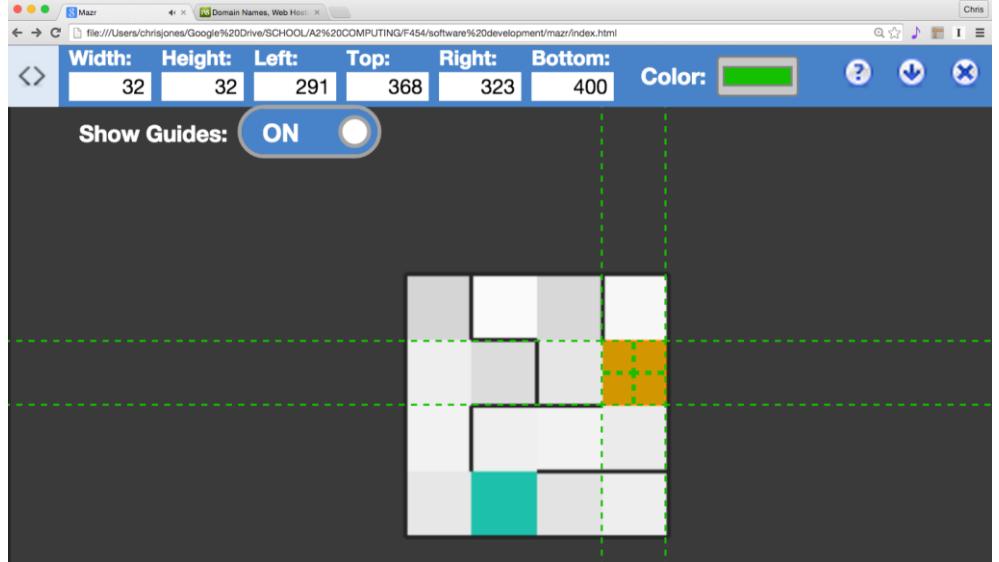
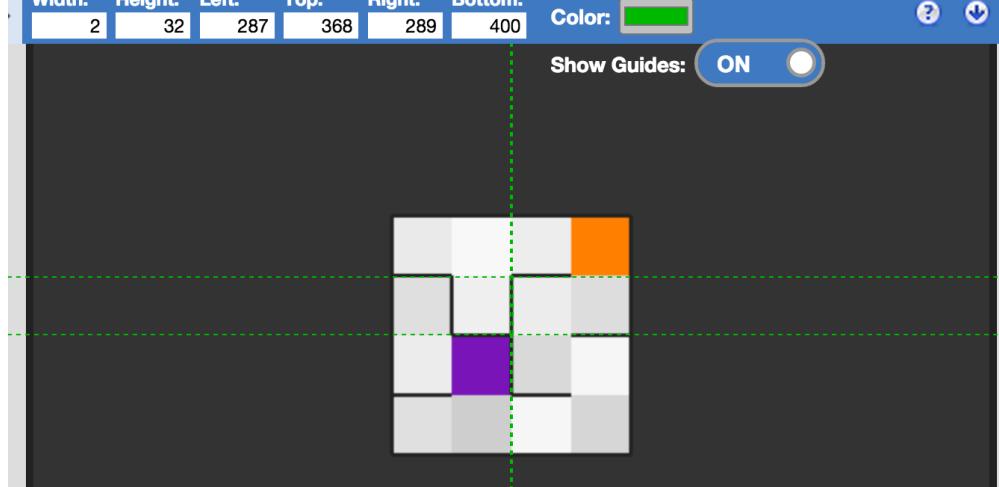
	
71	<p>The image below shows how a pixel ruler browser extension was used to measure the size of the game canvas. Note that the green overlay represents the area being measured. The width and height of this rectangle, placed over the game canvas, can be read off as 512 by 512 pixels. This is as intended.</p> 
72	Through basic visual inspection of a maze, shown in the image below, it can be determined that the tiles are drawn in a grid with up to four walls, including the player and exit tile.

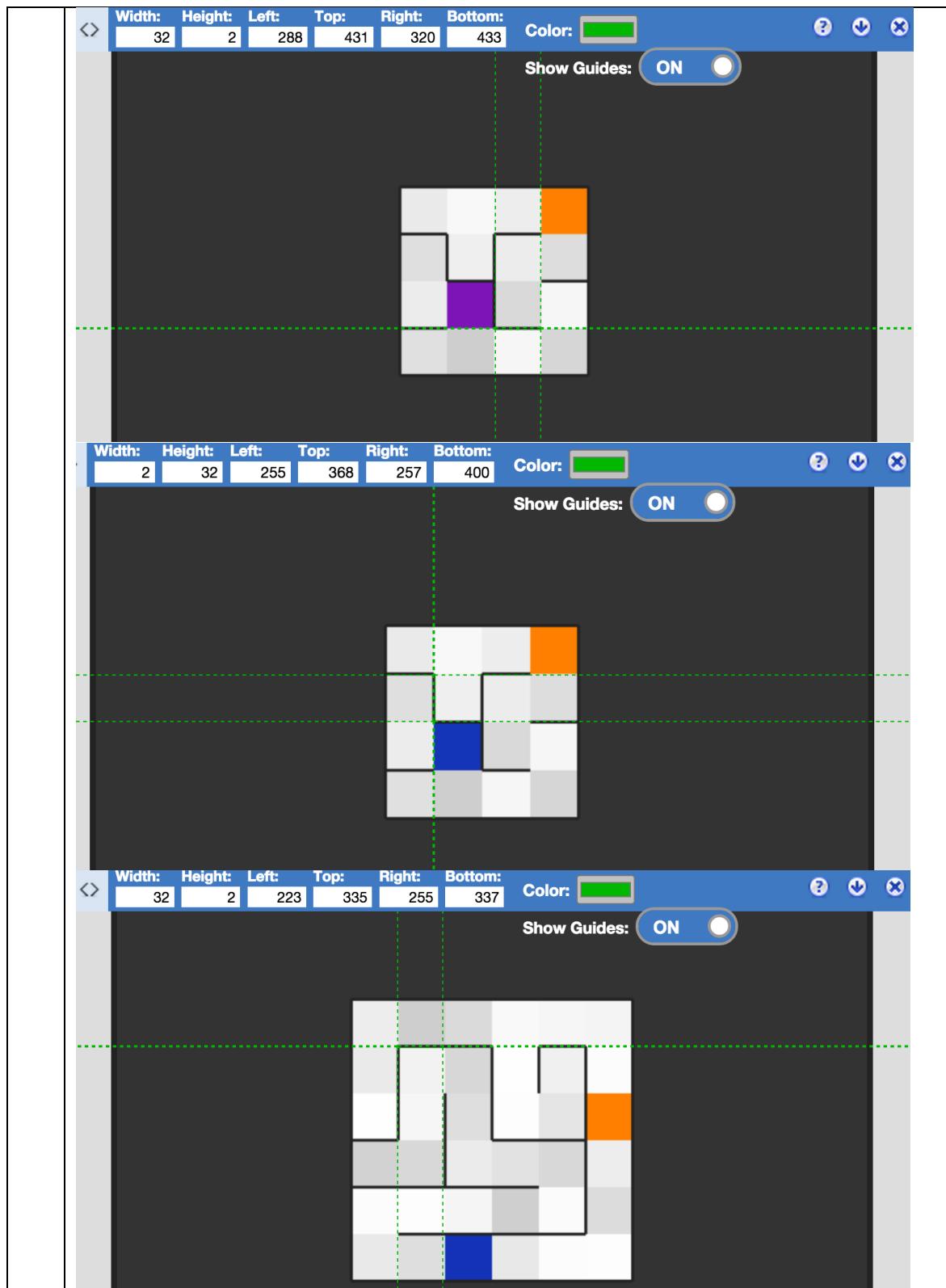
	<h1>Mazr</h1> <p>The endless maze solving puzzle game</p> 
73	<p>As it is unfeasible to test the size of every tile on every possible configuration of maze therefore to check for the correct size only a small sample have been selected and if they are the correct size it will be assumed all the tiles are the correct size.</p> <p>The following images show a pixel ruler browser plugin being used with a green overlay that is placed on top of the tiles in the maze and its size adjusted. The width and height in pixels is displayed on the bar in the top left corner. Through visual inspection it can be determined that in each image the tile being measured is 32 by 32 pixels. This is as intended.</p> 

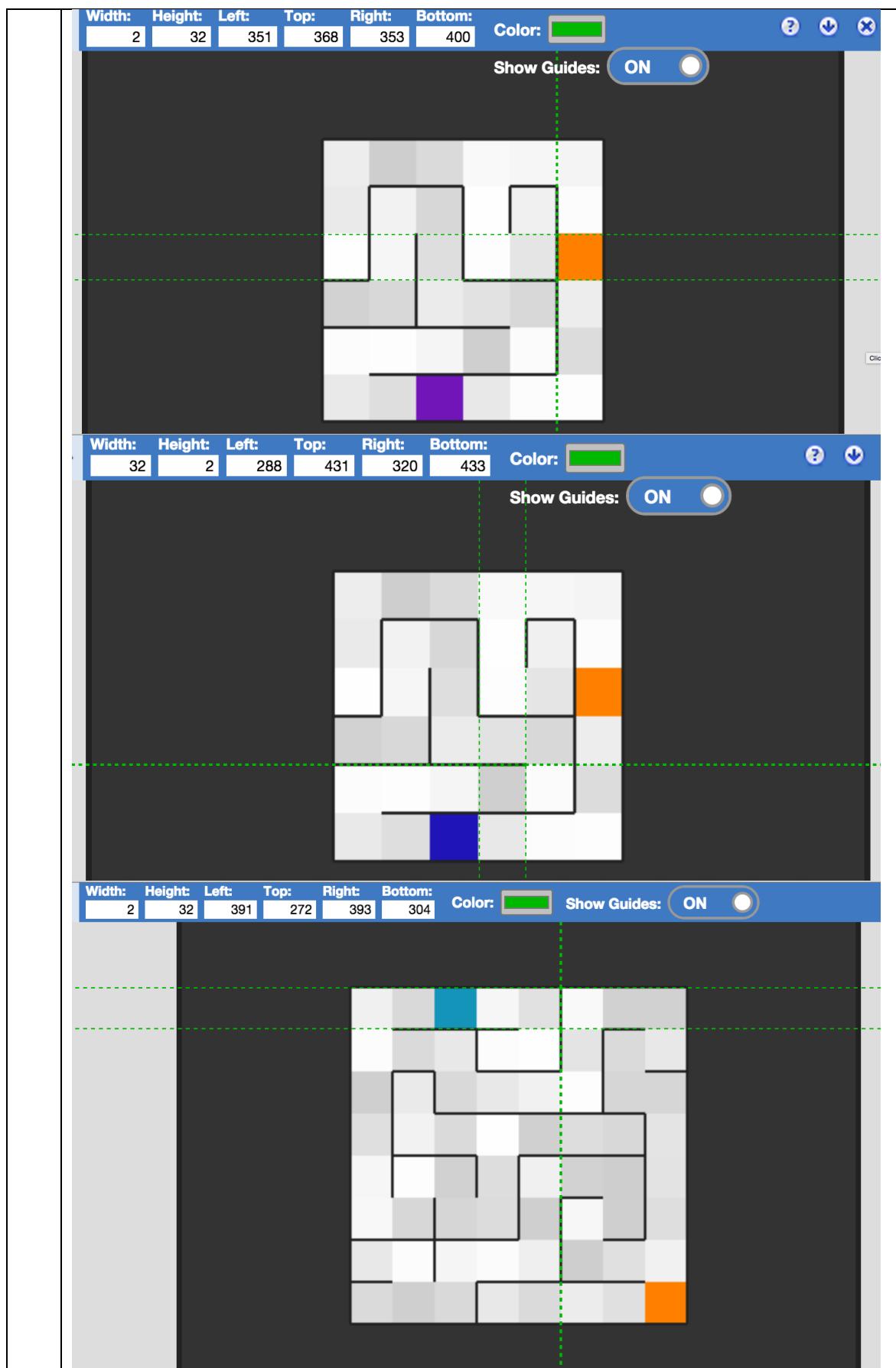


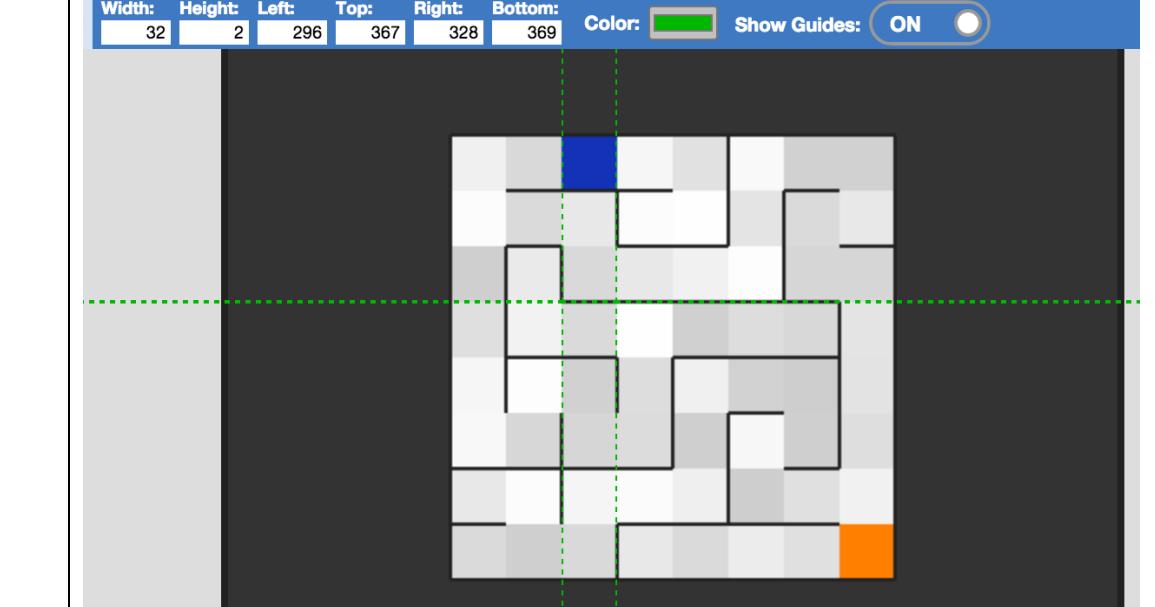
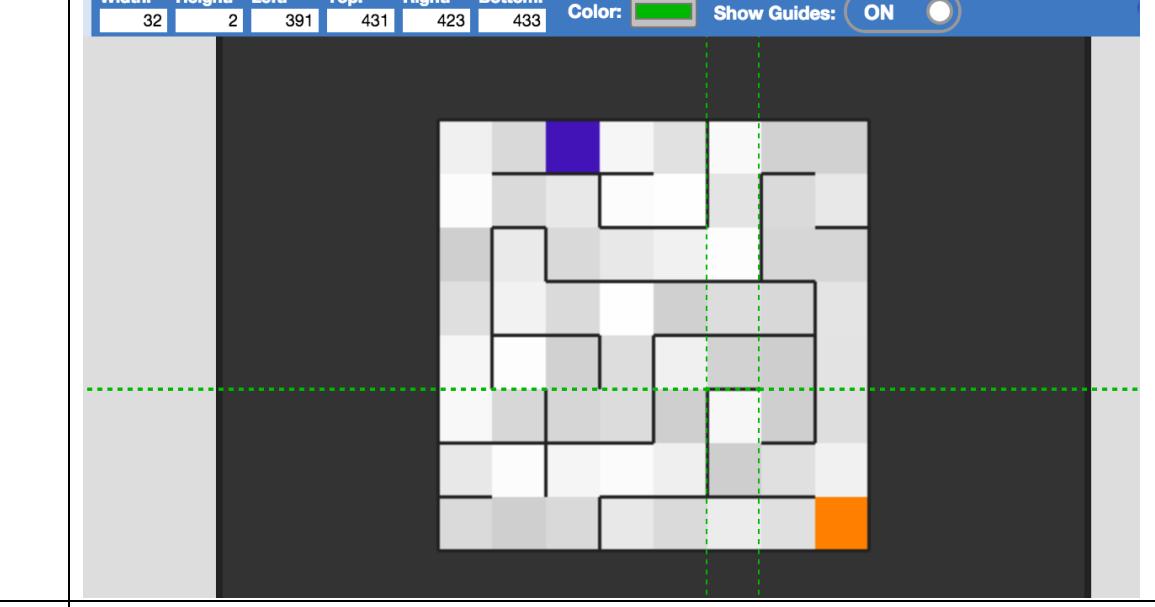
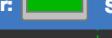


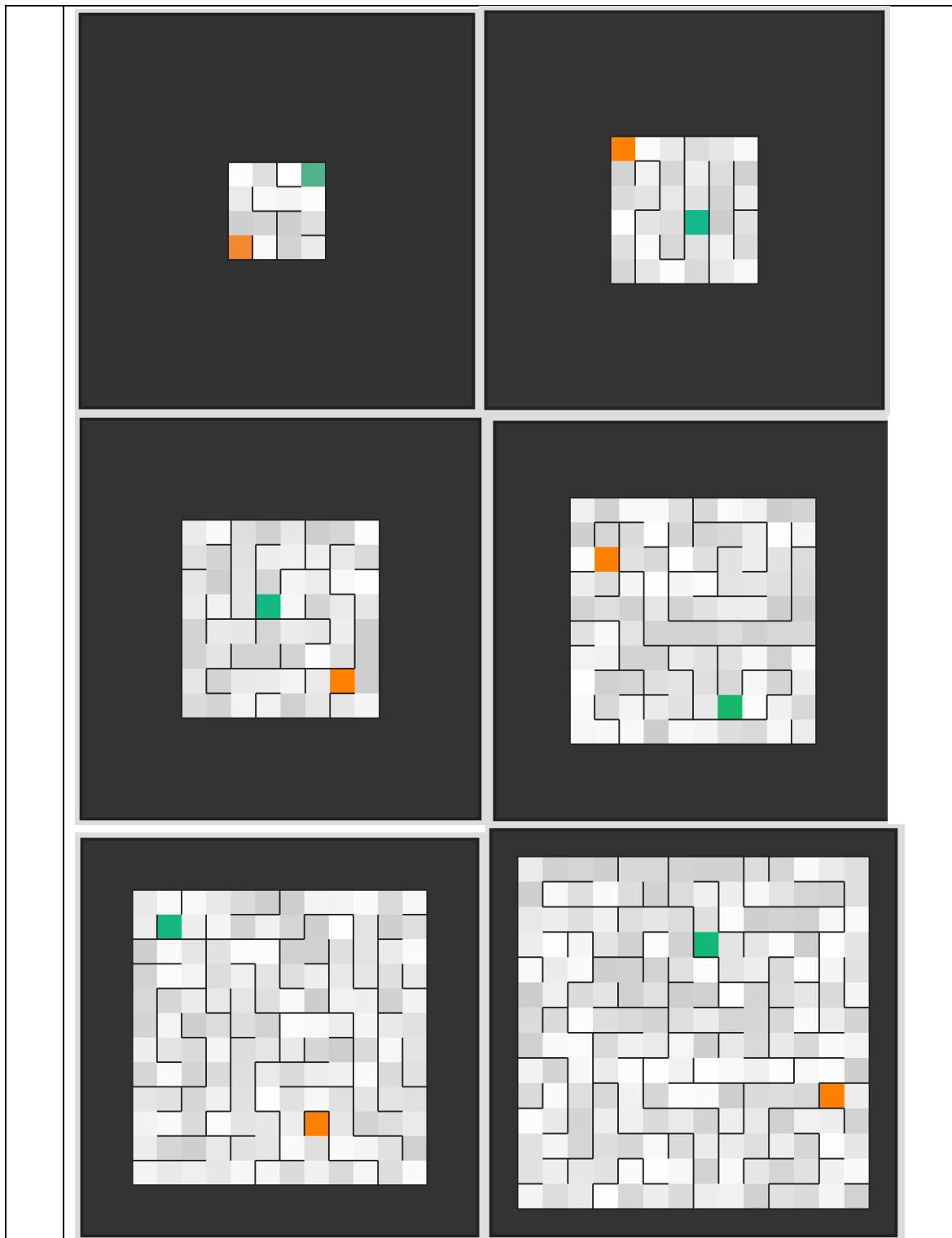
	 <p>A screenshot of a pixel art game titled "Mazr". A character is positioned in the center of a 32x32 grid. A green dashed rectangle highlights the character's bounding box. The top status bar shows: Width: 32, Height: 32, Left: 392, Top: 335, Right: 424, Bottom: 367, Color: green, Show Guides: ON.</p>  <p>The same screenshot as above, but the character has moved to the right. The green dashed rectangle now highlights the exit tile. The top status bar shows: Width: 32, Height: 32, Left: 423, Top: 432, Right: 455, Bottom: 464, Color: green, Show Guides: ON.</p>
74	<p>The image below shows a pixel ruler used to overlay a rectangle over the player. Note that the value of width and height of the rectangle overlay can be read off as 32 by 32 pixels. This is as intended.</p>  <p>A screenshot of a browser window titled "Mazr" showing the same 32x32 grid and character movement. The top status bar shows: Width: 32, Height: 32, Left: 228, Top: 432, Right: 260, Bottom: 464, Color: green, Show Guides: ON.</p>
75	<p>The image below shows a pixel ruler used to overlay a rectangle over the exit tile. Note that the value of width and height of the rectangle overlay can be read off as 32 by 32 pixels. This is as intended.</p>

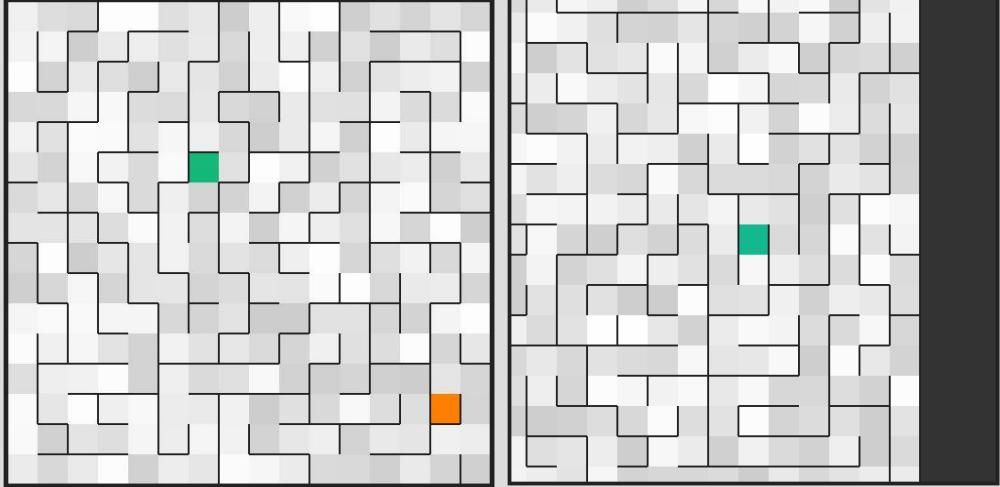
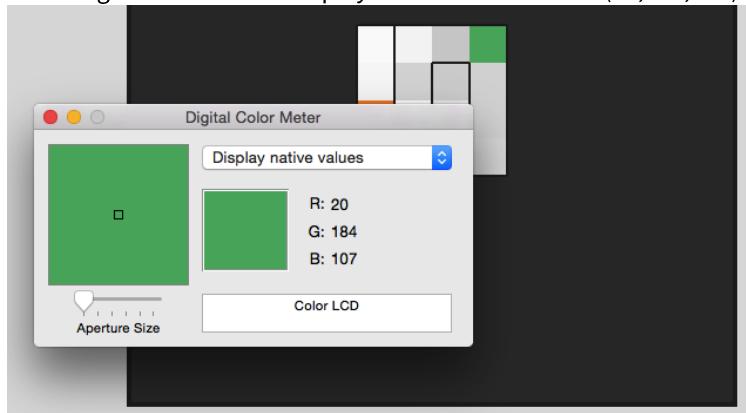
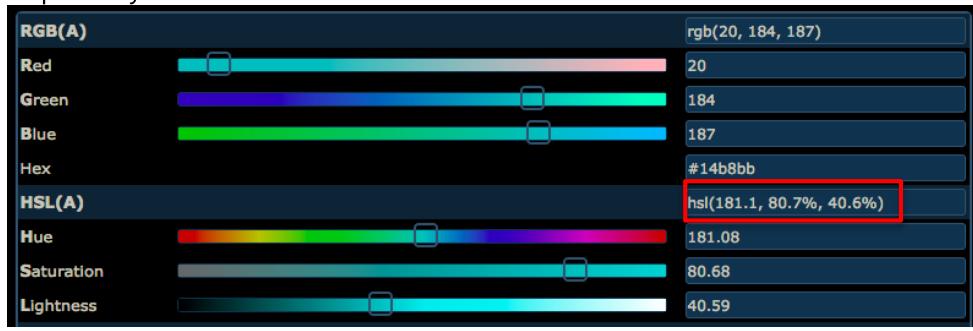
	 <p>Width: 32 Height: 32 Left: 291 Top: 368 Right: 323 Bottom: 400 Color: <span style="background-color: green; width: 20px; height: 10px; display: inline-block;"></span></p> <p>Show Guides: <b>ON</b></p>
76	<p>As it is unfeasible to test the size of every wall in every possible configuration of maze therefore to check for the correct size only a small sample have been selected and if they are the correct size it will be assumed all the walls are the correct size.</p> <p>The following images show a pixel ruler browser plugin being used with a green overlay that is placed on top of the walls in the maze and its size adjusted. The width and height in pixels is displayed on the bar in the top left corner. Through visual inspection it can be determined that in each image the wall being measured is 32 by 2 pixels or 2 by 32 pixels depending on whether the wall is horizontal or vertical. This is as intended.</p>  <p>Width: 2 Height: 32 Left: 287 Top: 368 Right: 289 Bottom: 400 Color: <span style="background-color: green; width: 20px; height: 10px; display: inline-block;"></span></p> <p>Show Guides: <b>ON</b></p>

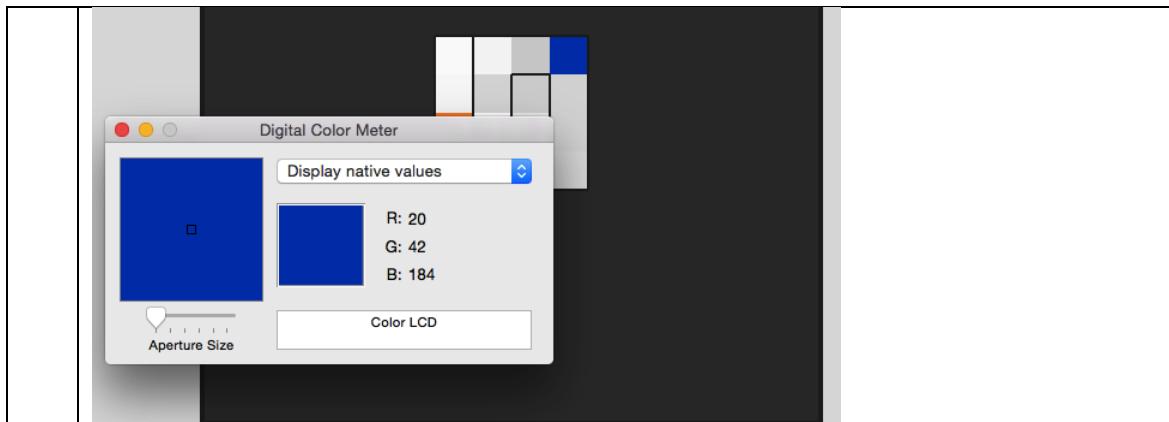




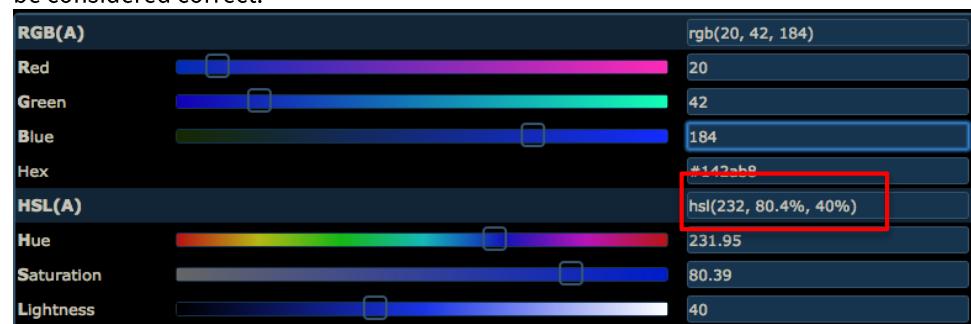
	 <p>Width: 32 Height: 2 Left: 296 Top: 367 Right: 328 Bottom: 369 Color:  Show Guides: <input checked="" type="button"/> ON</p>
	 <p>Width: 32 Height: 2 Left: 391 Top: 431 Right: 423 Bottom: 433 Color:  Show Guides: <input checked="" type="button"/> ON</p>
77	A sample of six different levels was taken and visually inspected to see if the tiles were a random assortment of the shades of grey. Indeed all of the mazes presented in the images below had tiles that were a random assortment of shades of grey (excluding the player and the exit tile of course).



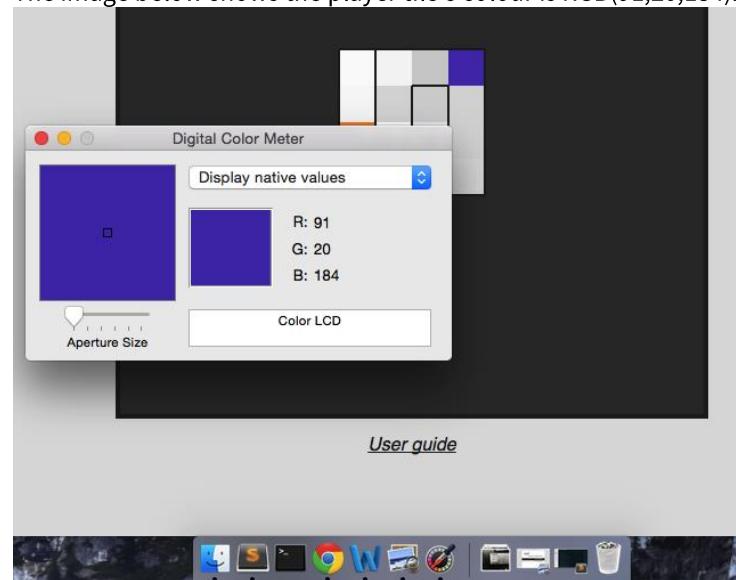
	
78	<p>This test will involve sampling the player's colour value four times over time and checking for constant values of saturation and lightness. To complete this test a two-step approach will be taken. As I could not find a HSL eyedropper tool that easily allowed me to sample the player colour I decided to use the native "Digital Color Meter" application on Mac OS X that would return an RGB value. I then entered this value into the colour picker on <a href="http://colorizer.org/">http://colorizer.org/</a>, which displayed the corresponding HSL value which could then be recorded.</p> <p>The image below shows the player tile's colour is RGB(20,184,107).</p>  <p>With the corresponding values entered into colorizer, the HSL output was (181.1, 80.7%, 40.6%), as shown in the image below. This is accurate enough to the values of 80% and 40% respectively to be considered correct.</p>  <p>The image below shows the player tile's colour is RGB(20,42,184).</p>



With the corresponding values entered into colorizer, the HSL output was (232, 80.4%, 40%), as shown in the image below. This is accurate enough to the values of 80% and 40% respectively to be considered correct.



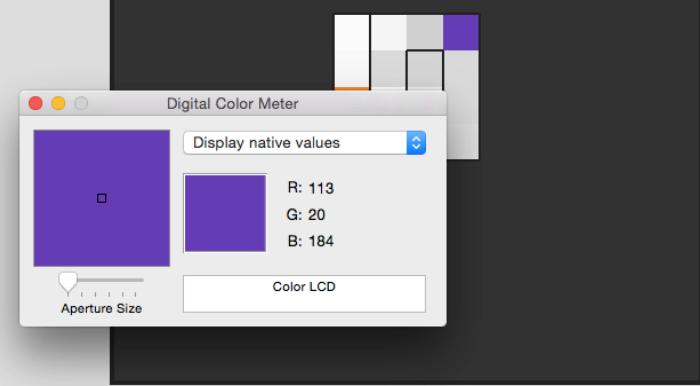
The image below shows the player tile's colour is RGB(91,20,184).



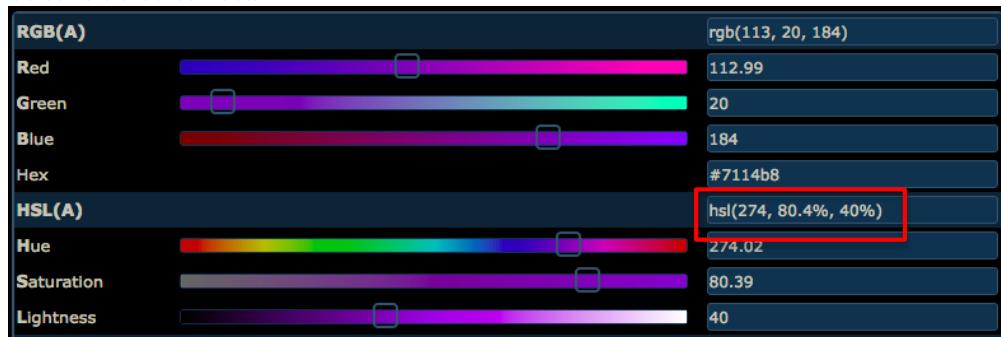
With the corresponding values entered into colorizer, the HSL output was (256, 80.4%, 40%), as shown in the image below. This is accurate enough to the values of 80% and 40% respectively to be considered correct.



The image below shows the player tile's colour is RGB(113,20,184).

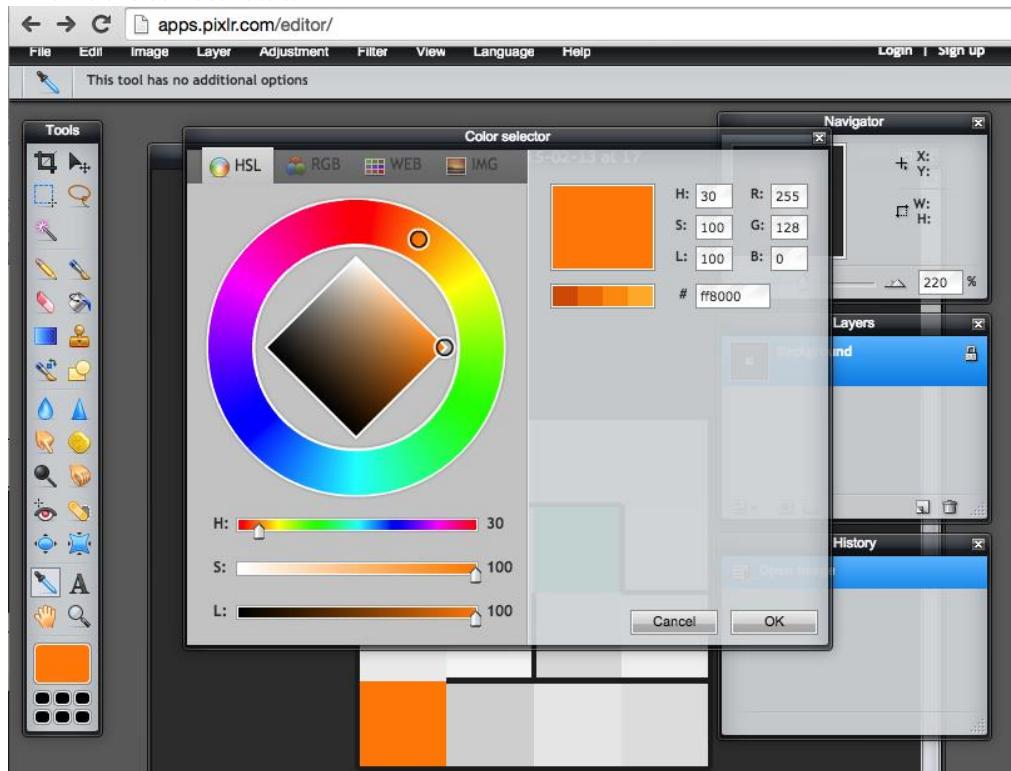


With the corresponding values entered into colorizer, the HSL output was (274, 80.4%, 40%), as shown in the image below. This is accurate enough to the values of 80% and 40% respectively to be considered correct.



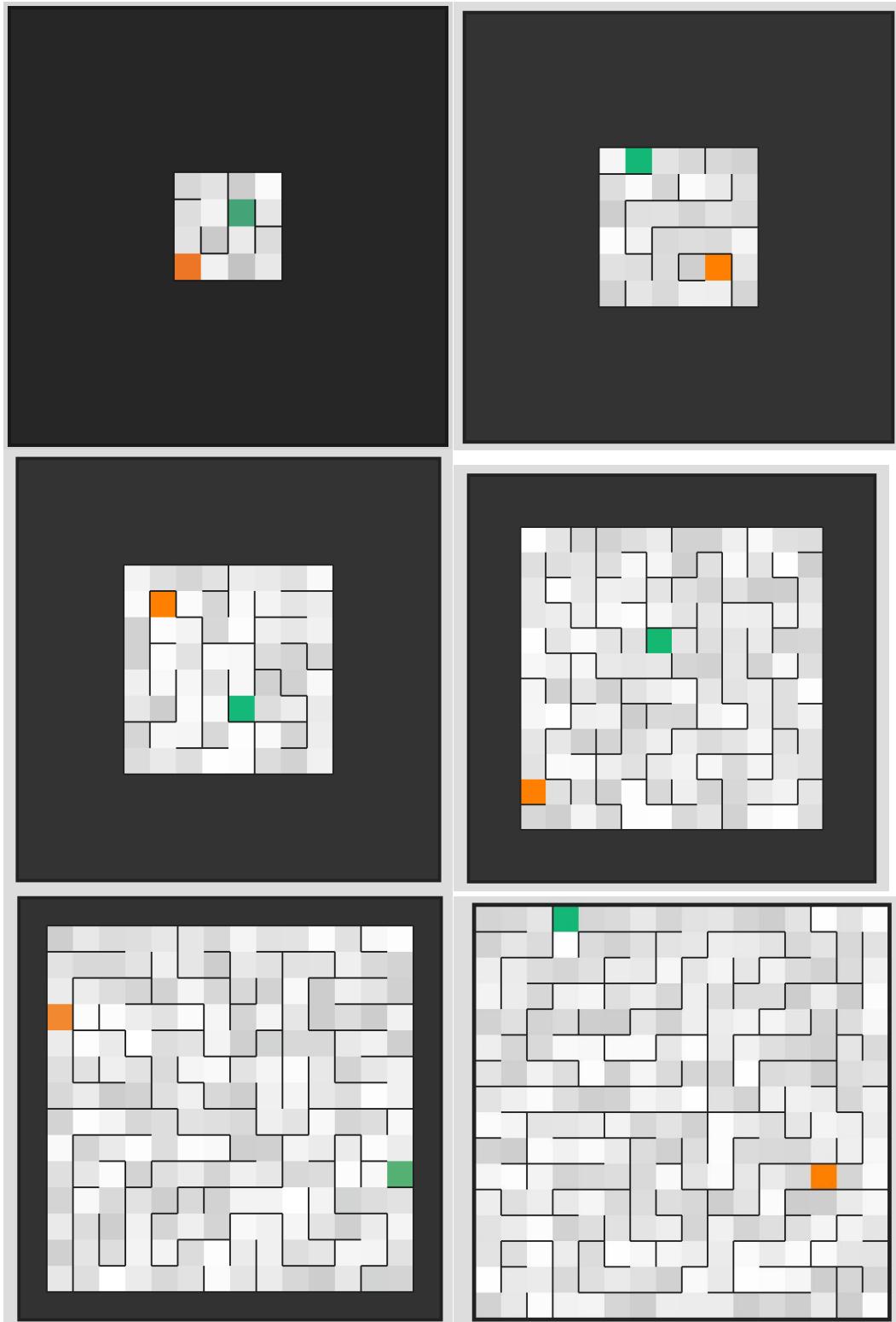
Therefore, with these four samples being shown as correct it can be considered the hue and saturation remains constant over time, as intended.

- 79 Image below shows the colour of the exit tile being determined with the eyedropper in the pixlr web app. The colour is shown in the 'Color selector' window below. Note the hex value #ff8000, which is the correct value.



- 80 The following six images show mazes that fit inside the canvas (it can easily be determined each

of these fits in the 512 by 512 pixel canvas as the maximum dimensions of a maze composed of 32 by 32 pixel tiles is  $512/32 = 16$  tiles in each dimension). Note that by visual inspection it can be determined that each of these mazes is centred in the canvas and by extension that the camera is disabled. This is as intended.



## Response to post development testing

This section is the response to the testing that has just been completed. Various tests were in fact unsuccessful and as such some necessary modification must take place in order for the solution to function correctly.

Below are details of all modifications that will take place in order to resolve any errors or failed tests. Note at the end of this section the tests that failed are repeated and their success reevaluated.

### Continue button enabling / disabling correctly

#### (Test 38 and 39)

When a quit to main menu button is pressed, either on the pause menu or end of level menu – the user is navigated to the main menu. The continue button should be enabled to then allow the player to resume a maze of correct dimensions. However, this button is still disable and unusable when the main menu is displayed.

In order to resolve this problem, I decided it would be beneficial to examine the code that controlled the disabled of the continue button. Therefore, I opened the engine.js file and located the navigateTo function.

```

88 // Navigate to page with HTML id pageString
89 function navigateTo(pageString) {
90     if (pageString == ".main-menu") {
91         // If the currently stored level is one
92         if (localStorage["level"] == 1) {
93             // Disable the continue button
94             $('.btn-continue').attr('disabled', 'disabled');
95             // Set the text within the button to 'Continue'
96             // This is incase it contains a level number
97             $('.btn-continue').html('Continue');
98         } else {
99             // Otherwise insert the level the player is currently on
100            $('.bt n-continue').removeAttr('disabled');
101            $('.btn-continue').html(
102                'Continue <br><p class="under-text">From level ' +
103                localStorage["level"] + "</p>";
104            );
105        }
106    }
107
108    // If navigating away from the canvas
109    if (pageStack[pageStack.length-1] == "#canvas") {
110        // Disable the player
111        maze.player.disabled = true;
112        // Switch the current state to menu
113        currentState = STATE.MENU;
114    }
115
116    // If navigating to the canvas
117    if (pageString == "#canvas") {
118        // Enable the player
119        maze.player.disabled = false;
120
121        // Switch the current state to game
122        currentState = STATE.MAZE;
123    }
124
125    // Show the page that is being navigated to
126    showPage(pageString);
127
128    // Push it to the top of the pageStack
129    pageStack.push(pageString);
130 }
```

I began to examine the logic and looked to line 100, where the disabled attribute is removed from the continue button, which is not functioning correctly. I noticed the string passed to the jQuery object is not typed correctly – it should be “btn-continue” but instead is “bt n-continue”. This would appear to be the cause of the problem. I corrected this string to “btn-continue” which is shown in the image below.

```

88 // Navigate to page with HTML id pageString
89 function navigateTo(pageString) {
90     if (pageString == ".main-menu") {
91         // If the currently stored level is one
92         if (localStorage["level"] == 1) {
93             // Disable the continue button
94             $('.btn-continue').attr('disabled', 'disabled');
95             // Set the text within the button to 'Continue'
96             // This is incase it contains a level number
97             $('.btn-continue').html('Continue');
98         } else {
99             // Otherwise insert the level the player is currently on
100            $('.btn-continue').removeAttr('disabled');
101            $('.btn-continue').html(
102                'Continue <br><p class="under-text">From level ' +
103                localStorage["level"] + "</p>";
104            );
105        }
106    }

```

With this modification the test can be performed again in order to evaluate whether the problem has been removed.

### **Deletion of local storage**

#### **(Test 37)**

Processing test 13 failed because when local storage is deleted, the game does not recognise it has been loaded and does not disable the continue button and displays the subtitle contains the string “continue from level undefined”.

In order to resolve this problem, I opened the JavaScript console and cleared local storage by calling the clear method on the localStorage object. To ensure that it had been correctly cleared, I entered “localStorage”, which returned an empty hash table. Then, using the property access operator, I called for the properties “level”, “x” and “y” to be returned respectively. Each call returned an undefined value. Therefore, there needs to be a check added to the code which ensures that an undefined is not able to output to the continue button’s subtitle.

```

Elements Network Sources Timeline Profiles Resources Audits Console
localStorage <top frame> ▾ □ Preserve log
> localStorage.clear()
< undefined
> localStorage
< Storage {length: 0}
> localStorage["level"]
< undefined
> localStorage["x"]
< undefined
> localStorage["y"]
< undefined
>

```

I opened engine.js and located the navigateTo function, which handles the disabling of the continue button. The start of this function is shown in the image below.

```

87
88 // Navigate to page with HTML id pageString
89▼ function navigateTo(pageString) {
90▼   if (pageString == ".main-menu") {
91     // If the currently stored level is one
92▼     if (localStorage["level"] == 1) {
93       // Disable the continue button
94       $('.btn-continue').attr('disabled', 'disabled');
95       // Set the text within the button to 'Continue'
96       // This is incase it contains a level number
97       $('.btn-continue').html('Continue');
98▼   } else {
99     // Otherwise insert the level the player is currently on
100    $('.btn-continue').removeAttr('disabled');
101▼    $('.btn-continue').html(
102      'Continue <br><p class="under-text">From level ' +
103      localStorage["level"] + "</p>" +
104    );
105  }
106}
107

```

On line 92, there needs to be an additional check as to whether level equals undefined. The || or operator is used to check whether the level property is 1 or undefined. If the level is 1 or undefined, then the if statement's contents is executed which disables the continue button. This is added and shown in the image below.

```

88 // Navigate to page with HTML id pageString
89▼ function navigateTo(pageString) {
90▼   if (pageString == ".main-menu") {
91     // If the currently stored level is one
92▼     if (localStorage["level"] == 1 || localStorage["level"] == undefined) {
93       // Disable the continue button
94       $('.btn-continue').attr('disabled', 'disabled');
95       // Set the text within the button to 'Continue'
96       // This is incase it contains a level number
97       $('.btn-continue').html('Continue');
98▼   } else {
99     // Otherwise insert the level the player is currently on
100    $('.btn-continue').removeAttr('disabled');
101▼    $('.btn-continue').html(
102      'Continue <br><p class="under-text">From level ' +
103      localStorage["level"] + "</p>" +
104    );
105  }
106}
107

```

With this modification completed, processing test 13 can now be repeated in order to re-evaluate for success.

### **Escape button when not on a maze / the pause menu**

#### **(Test 27)**

The escape button is to be used to toggle between the pause menu and displaying a maze. The escape button should not do anything when used on any other menu. However, on the help menus and end of level menu, the escape key will act as a back function, returning to the previous menu / display of maze.

In order to resolve this problem, I navigated in the engine.js file to the switch-case statement that handles the keyboard event. I specifically investigated case 27, which handles the pressing of the escape key.

The if statement checks the user is not on the pause menu and is on the canvas menu. If this is true then they are navigated to the pause menu. Otherwise, the back goBack function is called. It

becomes apparent there is a logical error – when not on the canvas then the goBack function is called upon pressing the escape button. This is both incorrect behaviour and the initial if statement contains a complete redundant expression (why evaluate the user is not on the pause menu when it is already checking that they are on the canvas for this to evaluate true). Therefore, this code needs to be modified.

```

252     // Escape key code
253     case 27:
254         // If the current page is not the pause menu
255         // and the current page is the game canvas
256         if (pageStack[pageStack.length-1] != ".pause-menu" &&
257             pageStack[pageStack.length-1] == "#canvas") {
258
259             // Navigate to the pause menu
260             navigateTo('.pause-menu');
261
262         } else {
263
264             // Go back a step in the stack of pages
265             goBack();
266
267         }
268     }
269 //
```

Below is an image of the modification. The statement that checks the user is on the pause menu is copied from line 256 to an else if statement that handles execution of the goBack function. Note it is changed from a not equal (!= symbol) logical comparison to a does equal comparison (== symbol).

```

252     // Escape key code
253▼   case 27:
254     // If the current page is not the pause menu
255     // and the current page is the game canvas
256▼   if (pageStack[pageStack.length-1] == "#canvas") {
257
258     // Navigate to the pause menu
259     navigateTo('.pause-menu');
260
261▼   } else if (pageStack[pageStack.length-1] == ".pause-menu") {
262
263     // Go back a step in the stack of pages
264     goBack();
265
266   }
267 }
268 //
```

With this modification complete the test can be repeated to reassess whether success has been achieved.

#### **JavaScript error message**

##### **(Test 63)**

This issue arises when the user does not have JavaScript enabled in their browser. The error message is displayed however each menu is displayed at once which is not correct behaviour. This is because, as JavaScript is disabled, the code that manages the hiding of pages is never executed.

Therefore, all menus and the game canvas should be hidden by default and then made visible at the start of execution of the JavaScript engine.js. This is because, if JavaScript is disabled, the statement that renders the menus visible will never execute, so therefore remain hidden correctly.

The container in the HTML that contains all of the menus and the canvas needs to be set so that it is not be displayed by default. Therefore an additional term is entered into the CSS. The following selector is entered into the stylesheet.css file and is shown below. This sets the section element (which contains all of the game elements) to not be displayed upon the webpage being loaded.

```
7  
8  section {  
9    display: none;  
10 }  
11
```

In engine.js, I navigated to the block of code that is labelled as initialisation. This is shown in the image below.

```
279 // INITIALISE THE MAIN GAME  
280 //-----  
281 // setInterval is a function that takes a function as parameter.  
282 // Takes time interval in milliseconds as second parameter  
283 // Calls the function passed first every period that  
284 // was passed as second parameter  
285 setInterval(function() { // Anonymous function ...  
286   // Check the game is in maze state before calling functions  
287   if (currentState == STATE.MAZE) {  
288     // Call the update method first  
289     GAME.update();  
290     // Call the draw method  
291     GAME.draw();  
292   }  
293   //console.log(maze.completedSeconds);  
294  
295 }, 1000/FPS);  
296  
297 // Navigate to the main menu  
298 navigateTo('.main-menu');  
299  
300 gameAudio.play();  
301 //-----
```

Before the call to the setInterval function, I entered a call to the CSS method of the jQuery object, calling the display property to be set to block (this renders the section visible). This is shown in the image below.

```
279 // INITIALISE THE MAIN GAME  
280 //-----  
281 // Display the section that contains the game  
282 $('section').css('display', 'block');  
283
```

This can now be re-tested in order to evaluate success after modifications.

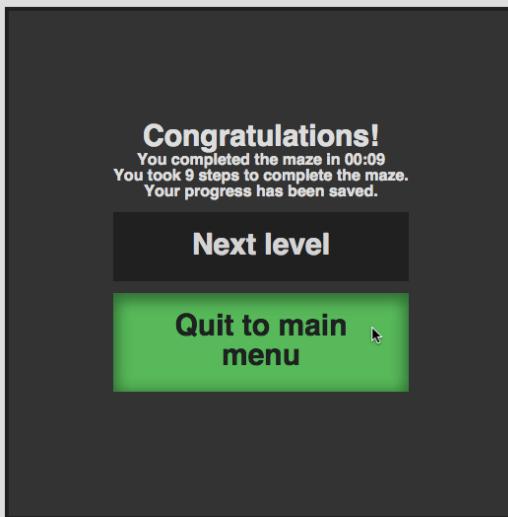
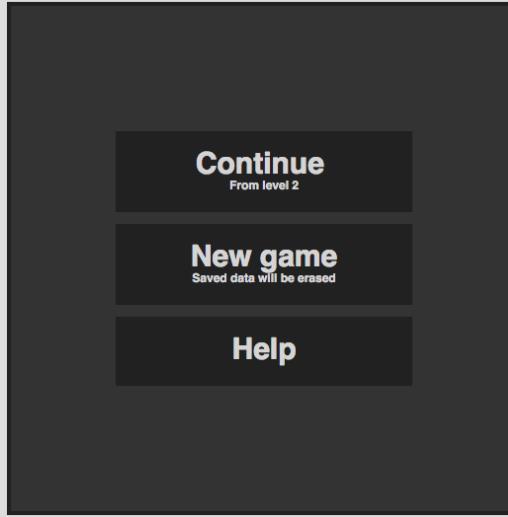
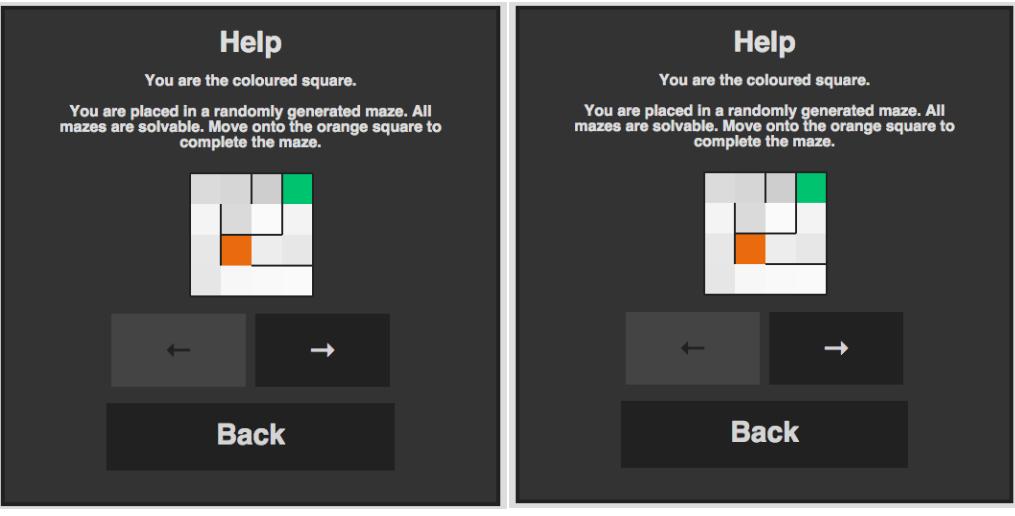
## Repeats of unsuccessful tests

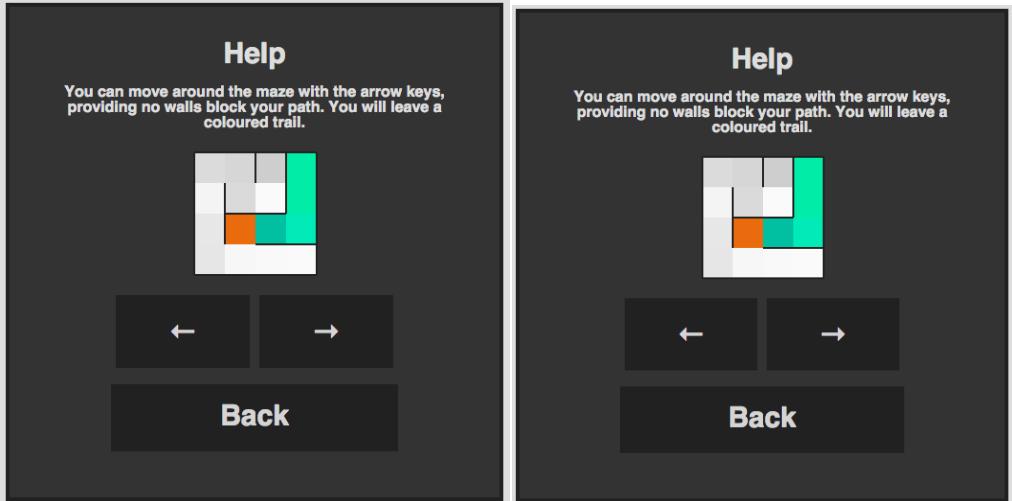
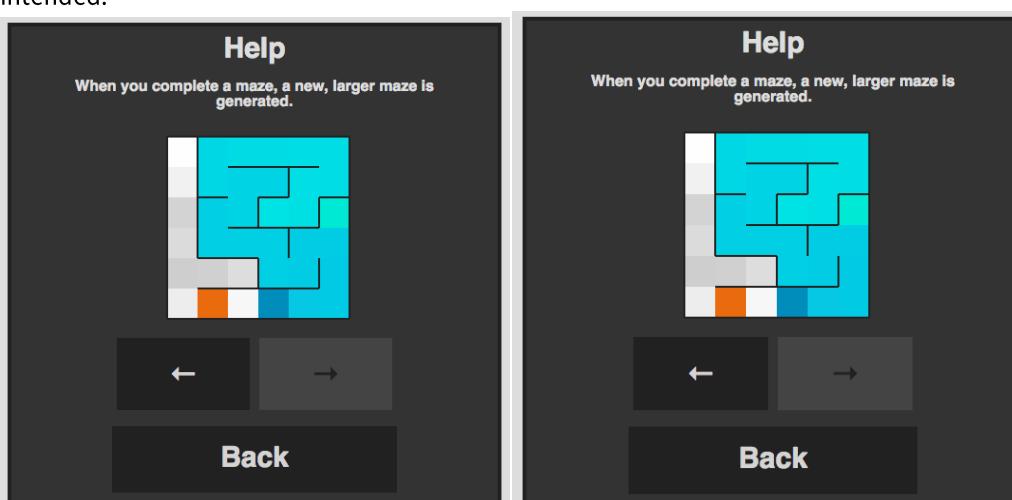
Test No	Objective Reference(s)	Purpose of test	Input / Test data	Expected Result	Actual Result (evidence)	Test successful?
37	2.c	Test that, if a game is saved in local storage and local storage is deleted, the game disables the continue button	Delete the contents of local storage in the JavaScript console.  Visit the webpage.	When local storage is deleted the continue button should be disabled.	Repeat Reference 1	Yes
38	2.c	Test that, when quitting to the main menu from the next level menu, that the continue button is enabled and the game can be loaded	Quit to main menu button (on end of level menu).  Continue button.	The continue button should not be disabled and when pressed should generate and display a maze of dimensions saved in local storage.	Repeat Reference 2	Yes
39	2.c	Test that, when quitting to the main menu from the pause menu, that the continue button is enabled and the game can be loaded.	Quit to main menu button (on end of level menu).  Continue button.	The continue button should not be disabled and when pressed should generate and display a maze of dimensions saved in local storage.	Repeat Reference3	Yes
27	1.b.ix	Test that pressing the escape key when on a help menu does nothing	Escape key	Nothing.	Repeat Reference 4	Yes
63	3.c	Test that when JavaScript is disabled in the user's browser, then a suitable error message is displayed and none of the menus are displayed.	N/A	A suitable error message should be displayed when the user has JavaScript turned off in their browser. No menus should be displayed.	Repeat Reference 5	Yes

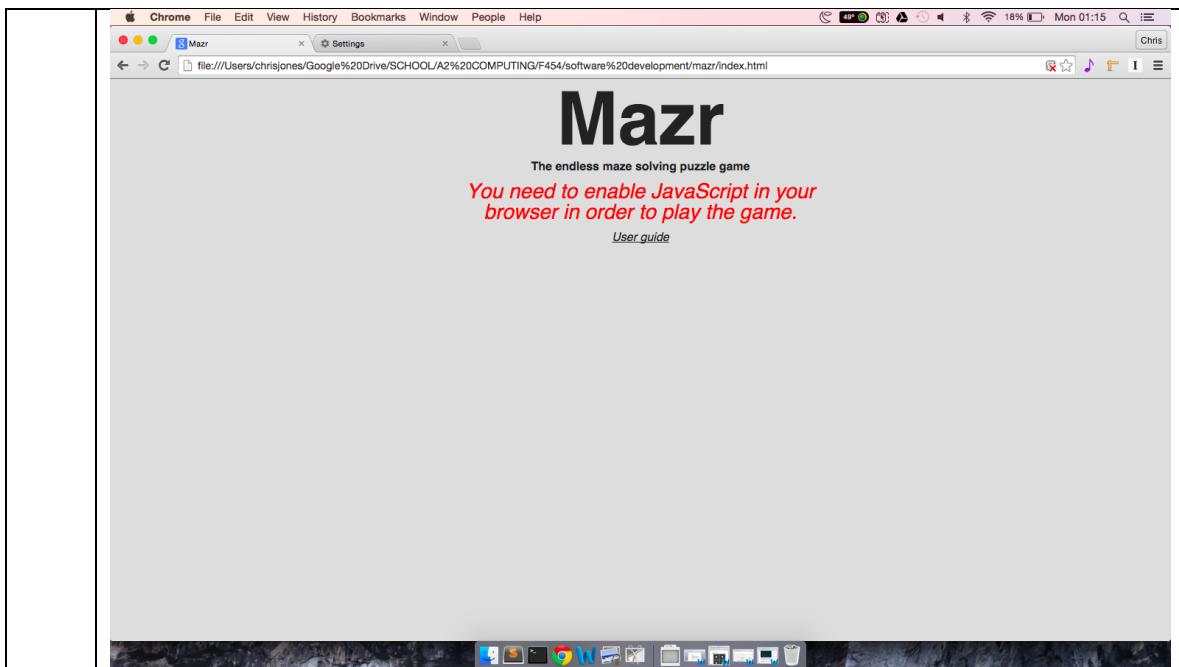
**Repeat test references**

Below is a table that contains the test evidence references of the results of the repeated tests. This is very similar in fashion to the references of the original run of tests.

Ref No	Reference contents
1	<p>Below is an image that shows the JavaScript console in a web browser. The command “localStorage” is entered to prove that there is data saved in Local Storage. Then the clear method is called on Local Storage with the “localStorage.clear()” command being entered. This deletes the contents of Local Storage. Another call to “localStorage” is made to determine that Local Storage no longer contains values.</p>  <pre> Elements Network Sources Timeline Profiles Resources Audits   Console &lt; top frame &gt; ▾ □ Preserve log &gt; localStorage &lt; Storage {level: "2", x: "6", y: "6", length: 3} &gt; localStorage.clear() &lt; undefined &gt; localStorage &lt; Storage {length: 0} &gt;   </pre> <p>Then, after a browser refresh, the main menu was displayed as shown below. The continue button is disabled, which is correct and desirable behaviour.</p> 
2	<p>The image on the left shows the Quit to main menu button on the end of level menu being pressed. The image on the right shows the main menu that is displayed once the button has been pressed. Note that the continue button is not disabled and displays a level from which to resume. This is intended behaviour.</p>

		
3	The image on the left shows the Quit to main menu button on the next level menu being pressed. The image on the right shows the main menu that is displayed once the button has been pressed. Note that the continue button is not disabled and displays a level from which to resume. This is intended behaviour.	
4	The left image below shows the first help menu. Upon the escape key being pressed, the first help menu is still displayed and no actions occur, as shown in the right image below. This is as intended.	

	<p>The left image below shows the second help menu. Upon the escape key being pressed, the second help menu is still displayed and no actions occur, as shown in the right image below. This is as intended.</p> 
	<p>The left image below shows the third help menu. Upon the escape key being pressed, the third help menu is still displayed and no actions occur, as shown in the right image below. This is as intended.</p> 
5	<p>The image below shows a selection in the settings of a web browser. Note that JavaScript has been disabled for all websites.</p> <p><b>JavaScript</b></p> <ul style="list-style-type: none"> <li><input type="radio"/> Allow all sites to run JavaScript (recommended)</li> <li><input checked="" type="radio"/> Do not allow any site to run JavaScript</li> </ul> <p><b>Manage exceptions...</b></p> <p>With JavaScript now disabled, the webpage on which the game is located was opened in a web browser, as shown in the image below. Note that an error message is displayed as intended and none of the menus are displayed. This is intended and desirable behaviour.</p>



## End user testing

### Introduction

The *Test Strategy* stipulated that end user involvement would be required to ensure that a thorough test of the system has taken place. Therefore a questionnaire for five end users to complete was drafted and presented in that section. In this section of *Testing*, that questionnaire can now be completed. The table is reproduced below and contains details

Question no	Question	Purpose of question	Frequency of response of users (Y/N)		Additional comments (optional)
			Yes	No	
1	Was the game easy to access via your web browser?	To gain an understanding of whether the game is accessible effectively in the end user's web browsers.	5	0	No comments.
2	Were the menus easy to use and understand?	Ascertain knowledge of whether the layout and content of the menus was easy to understand for the end users.	5	0	No comments.
3	Were the onscreen help menus sufficient and the user guide document sufficient in understanding how the game works?	To gain an understanding of whether the onscreen help menus that were integrated into the game contain sufficient information in order to play the game.	2	3	'The user guide is just a temporary PDF.'  'Incorrect user guide, just a placeholder. Wouldn't have needed it anyway as help menus are good enough'  'User guide link didn't link to a user guide.'
4	Was the system easy to use?	Ascertain knowledge of whether the end users find the game approachable and easy to use.	5	0	No comments.
5	Were the controls easy to use and suitable?	To gain an understanding as to whether the controls were natural for the end users or whether they had any trouble using them.	5	0	No comments.
6	Did the mazes provide a suitable challenge?	To understand whether the game provides a suitable challenge without being too easy or too difficult. Assessment of learning curve.	5	0	No comments.
7	Did you like the colour style and graphics used in the maze?	To ascertain whether the users like or dislike the ' <i>modern, blocky</i> ' style and graphics that are used in the game.	5	0	No comments.
8	Did you	To find out whether the users	1	4	'When I disabled

	discover any bugs or inconsistencies in the game? (Provide details in comments)	encountered any issues such as bugs or unexpected errors whilst playing the game.			setting data in Google Chrome the game would show all of the menus at once and none of the buttons work.'
9	Did the progress of the game save and load correctly?	To gain an understanding of whether the end users found that the saving and loading features worked when they played the game.	5	0	No comments.
10	Any further comments about potential modifications?	Enquire as to whether the end users had any further comments to make about the system and whether they have any ideas for modifications.	0	5	No comments.

### **Addressing the response to the end user testing**

Now that the end user testing has been completed there are a few issues that need to be addressed and resolved.

#### **Question 3**

There were multiple comments regarding the user guide link on the webpage redirecting the browser to a PDF file that contains a temporary empty file for user documentation. Note that this has not been forgotten and as such it will be replaced with the correct user documentation once it has been written. At this point nothing can really be resolved until the user documentation has been written and placed in the correct location.

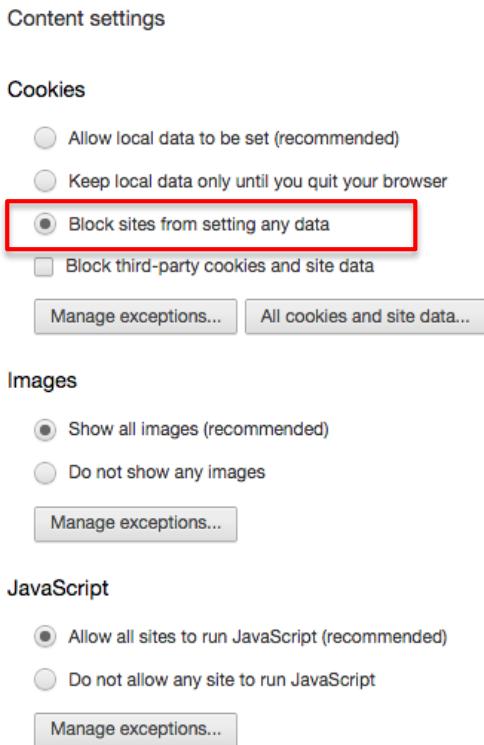
When the user documentation is written and placed in the correct location proof of this will be presented to the client and they will sign off to ensure that they agree the user documentation has been successfully completed.

**Question 8**

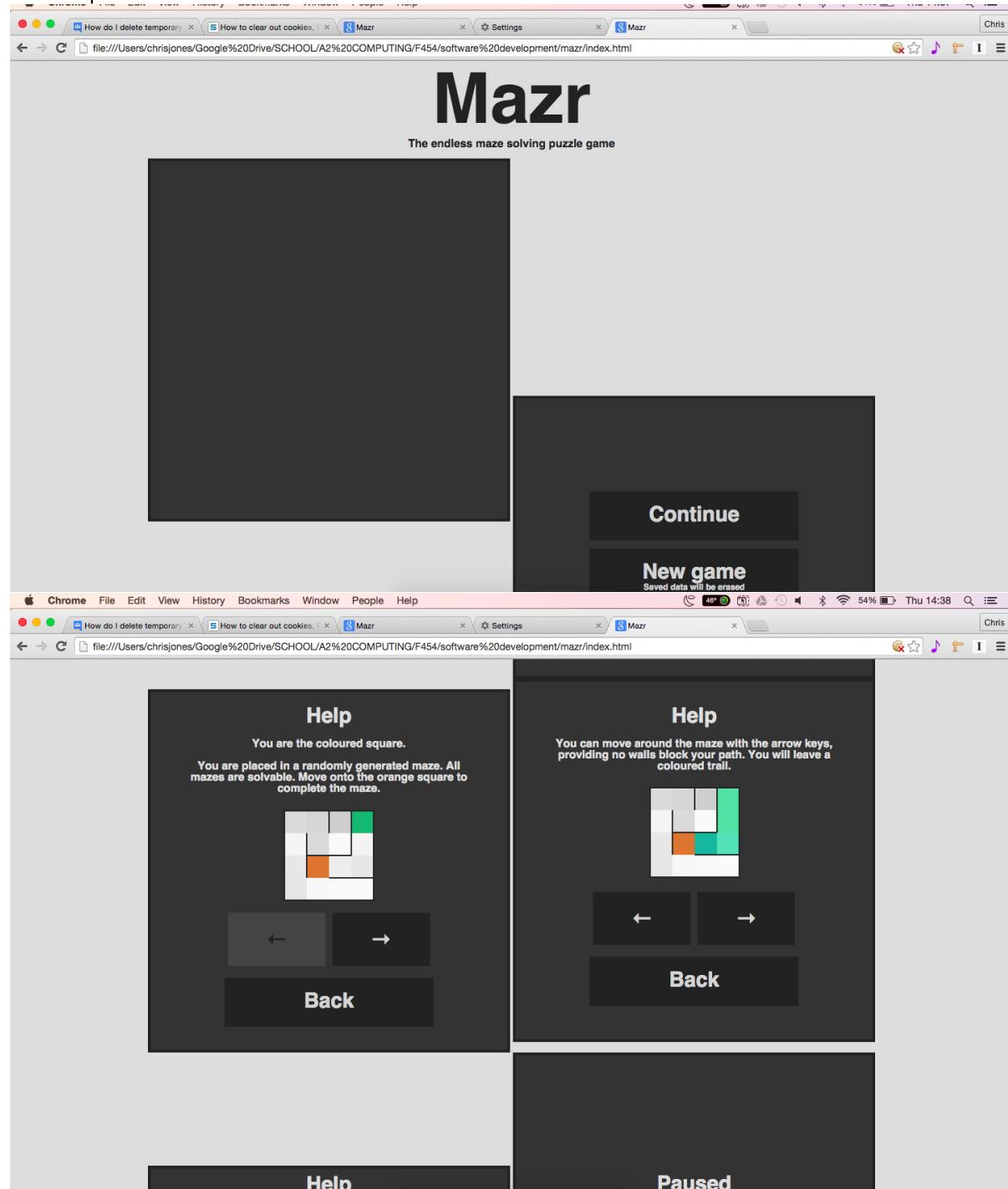
One of the users discovered a bug that they noted in question eight, which is produced below:

*'When I disabled setting data in Google Chrome the game would show all of the menus at once and none of the buttons work.'*

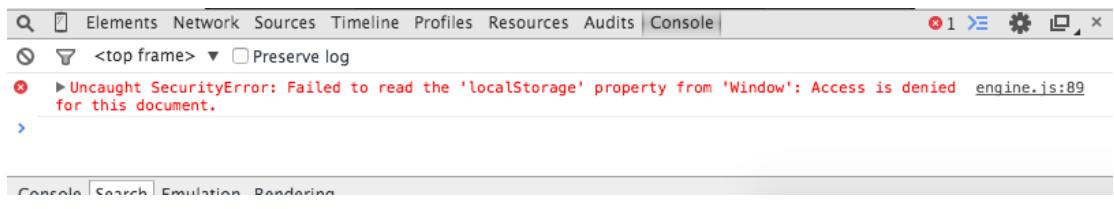
This seems like a bug relating to saving or loading of data in Local Storage. This is because it resulted from saving data being disabled. Therefore, I decided to attempt to reproduce the bug and test the conditions. In the Google Chrome web browser I blocked sites from setting any data. This is shown in the setting in the image below. Note that I used Google Chrome as the end user did but all browsers will have a similar setting that allows data setting to be disabled.



The two images below shows the result of visiting the webpage with the setting of data blocked. The canvas is inactive and none of the buttons are functional. This would suggest an error in the JavaScript that has halted execution.



I opened the JavaScript console in the web browser and found that indeed an error had been thrown. This is shown in the image below. It appears that this error is caused by attempting to access the Local Storage object on line 89.



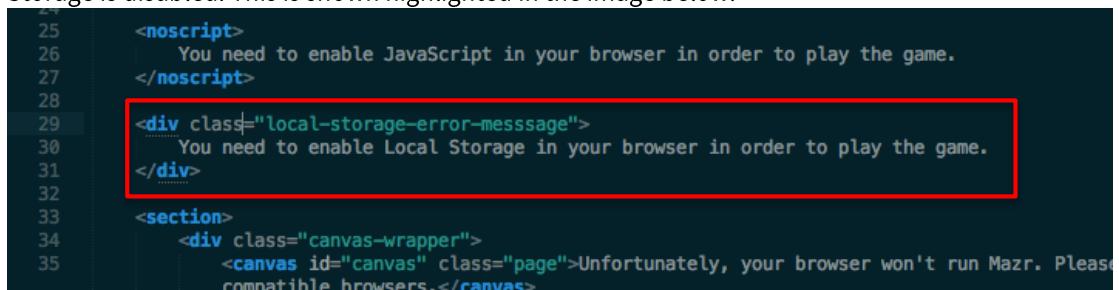
Note that on line 89, shown highlighted in the image below that the Local Storage object is attempted to access that must throw the error.

```

84
85 // Navigate to page with HTML id pageString
86 function navigateTo(pageString) {
87     if (pageString == ".main-menu") {
88         // If the currently stored level is one
89         if (localStorage["level"] == 1 || localStorage["level"] == undefined) {
90             // Disable the continue button
91             $('.btn-continue').attr('disabled', 'disabled');
92             // Set the text within the button to 'Continue'
93             // This is incase it contains a level number
94             $('.btn-continue').html('Continue');
95         } else {
96             // Otherwise insert the level the player is currently on
97             $('.btn-continue').removeAttr('disabled');
98             $('.btn-continue').html(
99                 'Continue <br><p class="under-text">From level ' +
100                 localStorage["level"] + "</p>";
101         );
102     }
103 }
104

```

To remedy this issue, a try and catch blocks will be added into the code. The try block will execute its contents and if an error is thrown then the catch block will be executed. Therefore, if a SecurityError is produced (as it is in this case) then the catch block will be executed when Local Storage access is attempted. The catch block will then hide all of the menus and display a suitable error. In the index.html file a div element is added that will display the error message when Local Storage is disabled. This is shown highlighted in the image below.



```

25 <noscript>
26     You need to enable JavaScript in your browser in order to play the game.
27 </noscript>
28
29 <div class="local-storage-error-message">
30     You need to enable Local Storage in your browser in order to play the game.
31 </div>
32
33 <section>
34     <div class="canvas-wrapper">
35         <canvas id="canvas" class="page">Unfortunately, your browser won't run Mazr. Please use compatible browsers.</canvas>

```

Then in the stylesheet.css file, the style is copied over so that it looks the same as the JavaScript error message – in order to provide some consistency. Note it is also set to display “none” and thus not be rendered when the style first loads.

```

11
12 noscript {
13   font-size: 28px;
14   font-weight: normal;
15   font-style: italic;
16   color: red;
17   width: 512px;
18   display: block;
19   margin: auto;
20 }
21
22 .local-storage-error-messsage {
23   font-size: 28px;
24   font-weight: normal;
25   font-style: italic;
26   color: red;
27   width: 512px;
28   display: block;
29   margin: auto;
30
31   display: none;
32 }
33

```

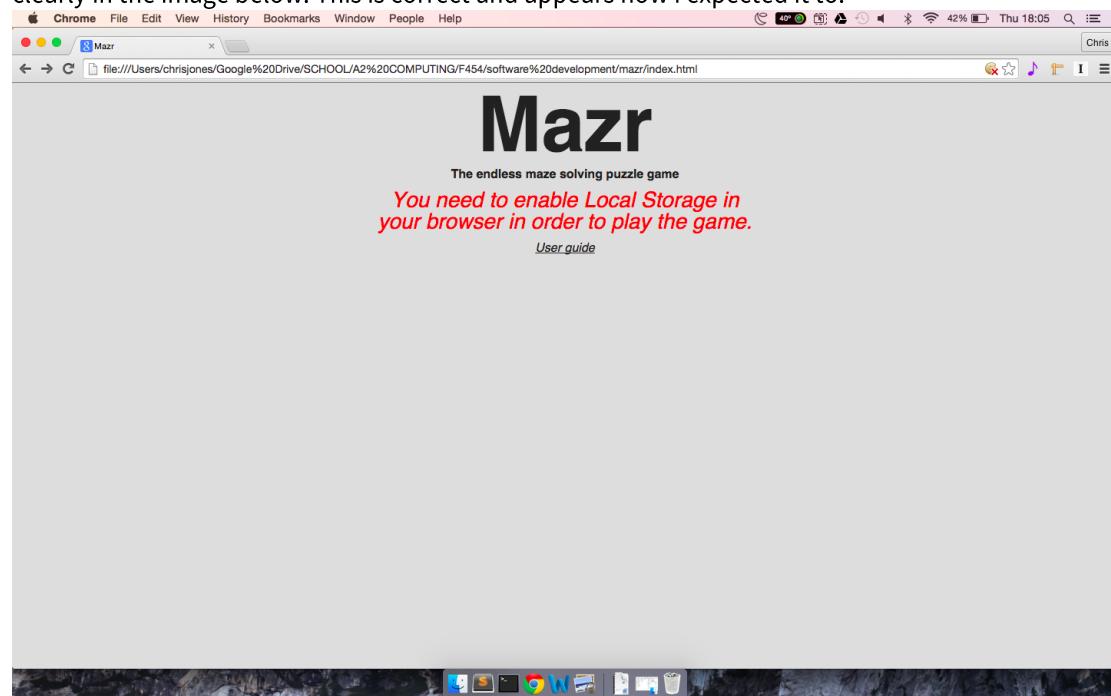
Then, the try statement is added to line 90 and the catch to line 108. Note that catch listens for the SecurityError and will execute its contents when one occurs. Therefore, if Local Storage settings are disabled then line 92 will throw an error. As it is inside the try block this will result in lines 110 and 113 being executing. This will result in the hiding of all the menus and displaying of the error message.

```

85 // Navigate to page with HTML id pageString
86 function navigateTo(pageString) {
87   if (pageString == ".main-menu") {
88     // This clause attempts to try the block of code
89     // Any SecurityErrors will be caught in the catch block
90     try {
91       // If the currently stored level is one
92       if ([localStorage]["level"] == 1 || [localStorage]["level"] == undefined) {
93         // Disable the continue button
94         $('.btn-continue').attr('disabled', 'disabled');
95         // Set the text within the button to 'Continue'
96         // This is incase it contains a level number
97         $('.btn-continue').html('Continue');
98     } else {
99       // Otherwise insert the level the player is currently on
100      $('.btn-continue').removeAttr('disabled');
101      $('.btn-continue').html(
102        'Continue <br><p class="under-text">From level ' +
103        [localStorage]["level"] + "</p>";
104      );
105    }
106  }
107  // This clause executes when SecurityError is thrown in the game
108  catch(SecurityError) {
109    // Hide the section
110    $('#section').css('display', 'none');
111
112    // Display the error message
113    $('.local-storage-error-messsage').css('display', 'block');
114  }
115}
116

```

With the setting not to allow data to be saved, I refreshed the web browser. The error is shown clearly in the image below. This is correct and appears how I expected it to.



One final check before proceeding is to ensure that once the setting has been switched off that the system functions as it did before the modification. The image below shows the setting has returned to default and data is allowed to be saved.

#### Content settings

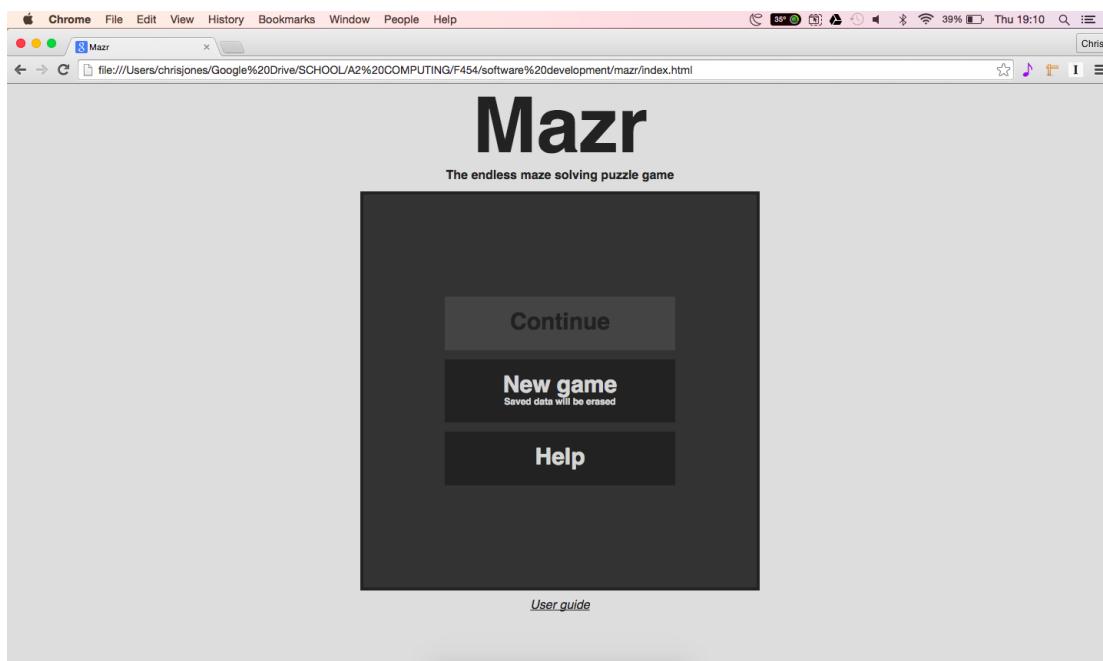
##### Cookies

- Allow local data to be set (recommended)
- Keep local data only until you quit your browser
- Block sites from setting any data
- Block third-party cookies and site data

[Manage exceptions...](#)

[All cookies and site data...](#)

Upon again refreshing the webpage, the menus are rendered as they were before. Upon some initial exploration it appears as if everything functions correctly. This is expected and indeed desirable behaviour. The main menu is shown rendered correctly in the image below.



## Acceptance Testing

### Introduction

This is the final stage of testing of the new system and requires the client's involvement in examining multiple stages of the Testing and signing off that they agree the system has been thoroughly tested and functions as was specified in the Design Specification.

The client was given copies of the test table, including all of the references associated, evidence of modifications pertaining to the unsuccessful tests and a table of successful repeats of the unsuccessful test along with associated evidence.

A short interview was conducted with the client to determine whether they had any additional comments or changes that were desired.

*Me:* Thank you for attending this short interview.

*Client:* That's okay.

*Me:* Did you get a chance to read the post development test table and the associated references?

*Client:* Yes I did. I looked at each of the tests and the references one by one to make sure I was satisfied Mazr worked. I was concerned at first because I noticed a number of unsuccessful tests but I realised you'd made changes to fix these issues.

*Me:* Oh yes any of the tests that were unsuccessful involved some modifications to the code taking place and then the test being repeated.

*Client:* Yes, I reviewed the [5] unsuccessful tests and saw they were repeated. This was pleasing as you included references so I could see you've got proof the tests pass.

*Me:* Are there any issues at all you have with this testing?

*Client:* No, I believe everything is fine and correct.

*Me:* Okay, that's good. Did you get a chance to review the User testing section?

*Client:* Yes. The comments were pleasing. The users seemed to enjoy the game and its features.

*Me:* Did you see the modification that was made with respect to one of the user's comments?

*Client:* Yes I saw that there was an error with saving data that you'd fixed. This would be good and I'm happy you went ahead and did that.

*Me:* So you had a chance to have a play with the game and test the system for yourself?

*Client:* Why yes, I did have a go at the game. It was very good and very faithful to the designs and screen layouts we agreed to early on in the project. The ease of access and controls was excellent and the mazes all seemed to be very random. I liked it a lot. However I noticed that the users reported in the end user testing that the manual linked to on the webpage links to the temporary file [shown in the Testing References]. I checked this myself when I used the system and it's still the same. Is this a mistake? Does it need to be changed?

*Me: The user documentation hasn't yet been written. It's planned to be written after this Testing stage. Once its completed I'll place it in the development folder.*

*Client: That's fine then.*

*Me: Are there any final modifications you wish to suggest before signing off on the testing?*

*Client: No. I am very happy with the game as it is now and nothing else needs to be changed.*

*Me: Excellent. I'll need you to sign to say that you have reviewed the testing documents and are happy that this stage of development is complete and the game fulfils the design specification.*

*Client: I would be happy to do so.*

*Me: Good. Next, I will be working on the user documentation for the game to replace the temporary file.*

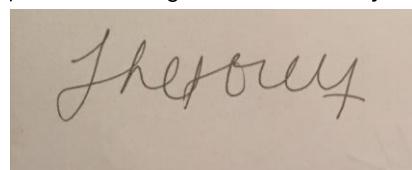
*Client: That's good to hear.*

*Me: We will need to arrange another interview in order to discuss an evaluation of the new system. We can discuss the requirements and the design objectives in detail.*

*Client: Yes, I'm happy to arrange something when you're ready.*

#### **Sign off of testing by client**

*The signature below certifies that the client has reviewed the testing (including references), modifications and retesting (including references), user testing and the client has reviewed the game and any modifications have been made that she is happy that the testing is exhaustive enough to prove that the game meets the objectives in the design specification.*

A handwritten signature in black ink on a light-colored rectangular card. The signature reads "Christopher Jones".

# Mazr

## User documentation

### Table of Contents

<b>Introduction.....</b>	<b>2</b>
<b>System Requirements .....</b>	<b>2</b>
<b>Software Requirements.....</b>	<b>2</b>
<b>Hardware Requirements .....</b>	<b>2</b>
<b>Accessing the game.....</b>	<b>3</b>
<b>Instructions on how to play MAZR .....</b>	<b>4</b>
<b>Starting a new game .....</b>	<b>4</b>
<b>Solving mazes .....</b>	<b>5</b>
<b>Solving larger mazes.....</b>	<b>7</b>
<b>Loading a saved maze .....</b>	<b>9</b>
<b>Viewing the help menus .....</b>	<b>10</b>
<b>Using the pause menu .....</b>	<b>11</b>
<b>Errors and troubleshooting .....</b>	<b>12</b>
<b>Accessing the game.....</b>	<b>12</b>
<b>Playing the game.....</b>	<b>12</b>
<b>Glossary.....</b>	<b>13</b>

## Introduction

This document contains all the information required to play Mazr – a puzzle game that requires the solving of mazes randomly generated by a computer.

## System Requirements

Although the game may run on hardware or software that is not mentioned in this section, it may not directly supported, and so the game may behave unpredictably. This is not recommended.

### Software requirements

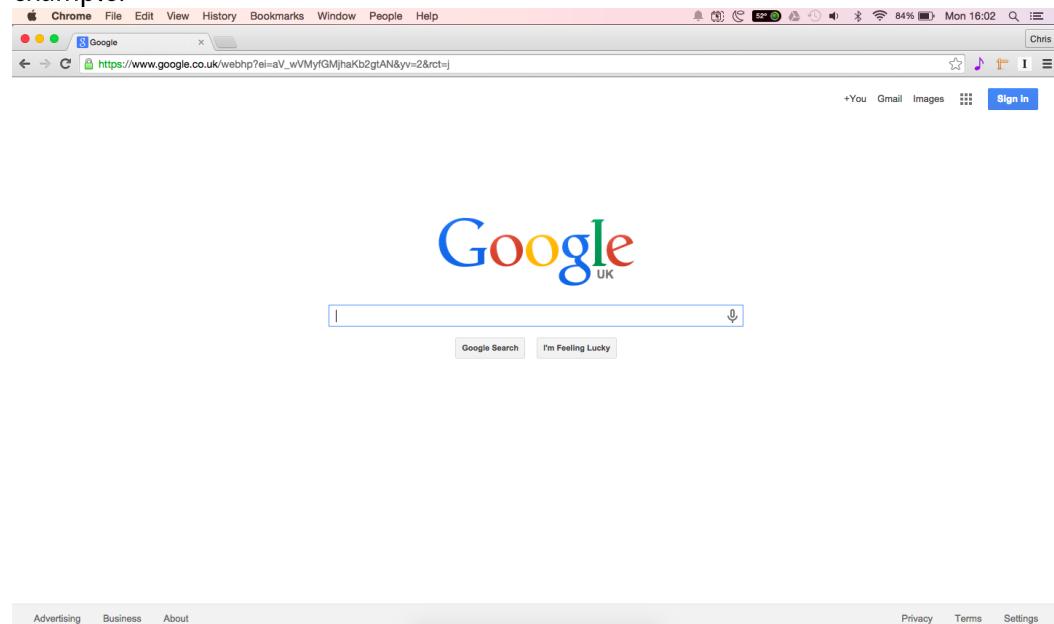
- One of the following operating systems must be installed and operational on your computer
  - Windows 7 or later
  - Mac OS X Lion or later
- One of the following web browsers to access the game:
  - *Google Chrome versions 27 and later*
  - *Internet Explorer versions 9 and later*
  - *Mozilla Firefox versions 30 and later*
  - *Safari versions 5.1 and later*
- JavaScript must be enabled in web browser.

### Hardware Requirements

- Internet connection.
- 5MB of storage space on computer.
- Keyboard connected to computer.
- A mouse / touchpad connected to computer.
- A monitor with a minimum resolution of 1024x768 or equivalent connected to computer.
- Must have 512MB of RAM
- Must have 1GHz or faster CPU with 32 bit or 64 bit architecture.

## Accessing the game

1. Open the web browser you're using to access the game. This must be one listed in the software requirements above. In the image below Google Chrome is being used as an example.

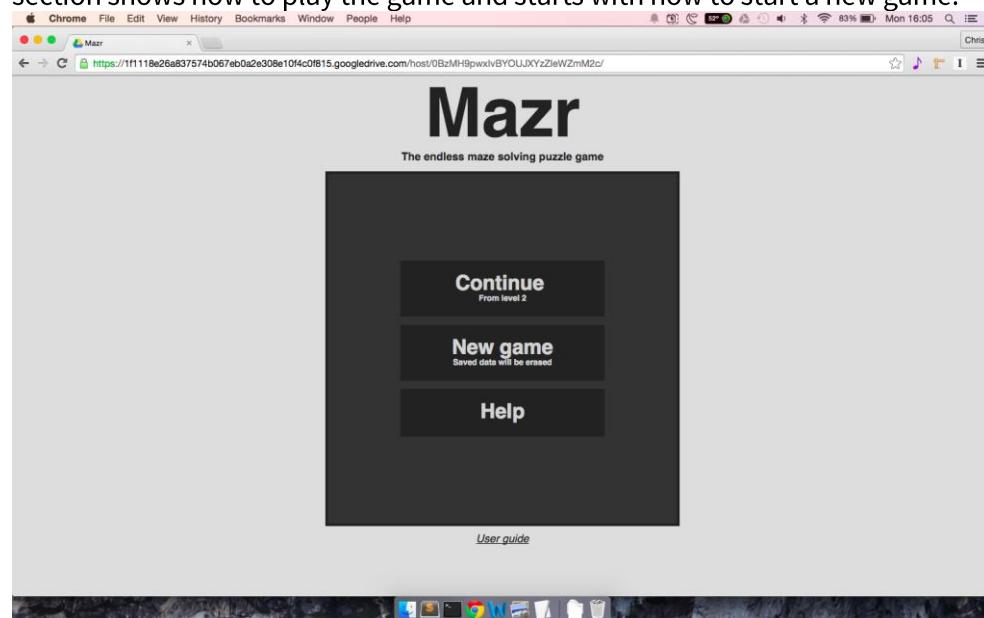


2. Enter the following web address into the URL bar of the web browser, as shown in the image below:

<http://bit.ly/1APzjl4>



3. The web browser will redirect to the webpage on which the game is hosted. The next section shows how to play the game and starts with how to start a new game.

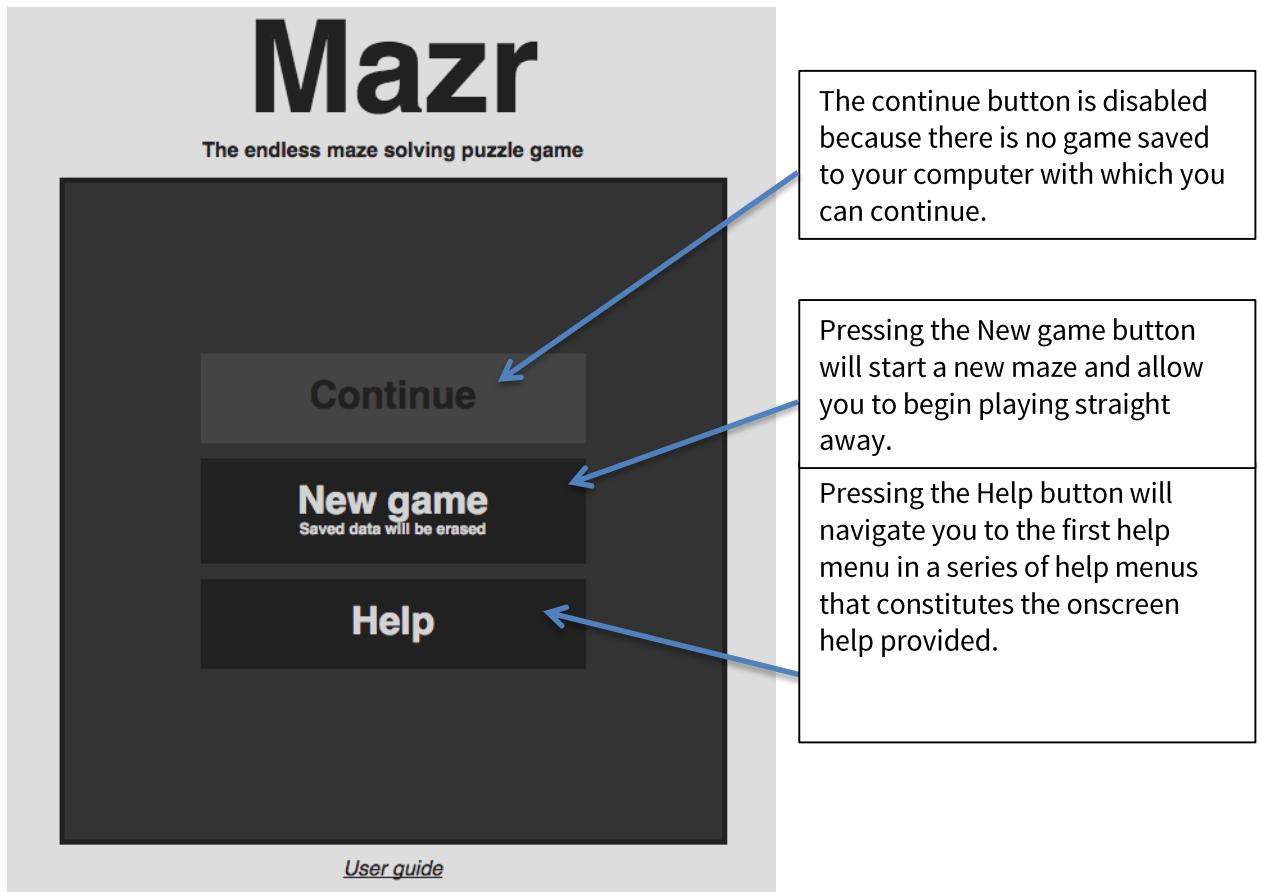


## Instructions on how to play Mazr

Mazr is a game with simple controls and yet provides a challenge that will require problem-solving abilities.

### Starting a new game

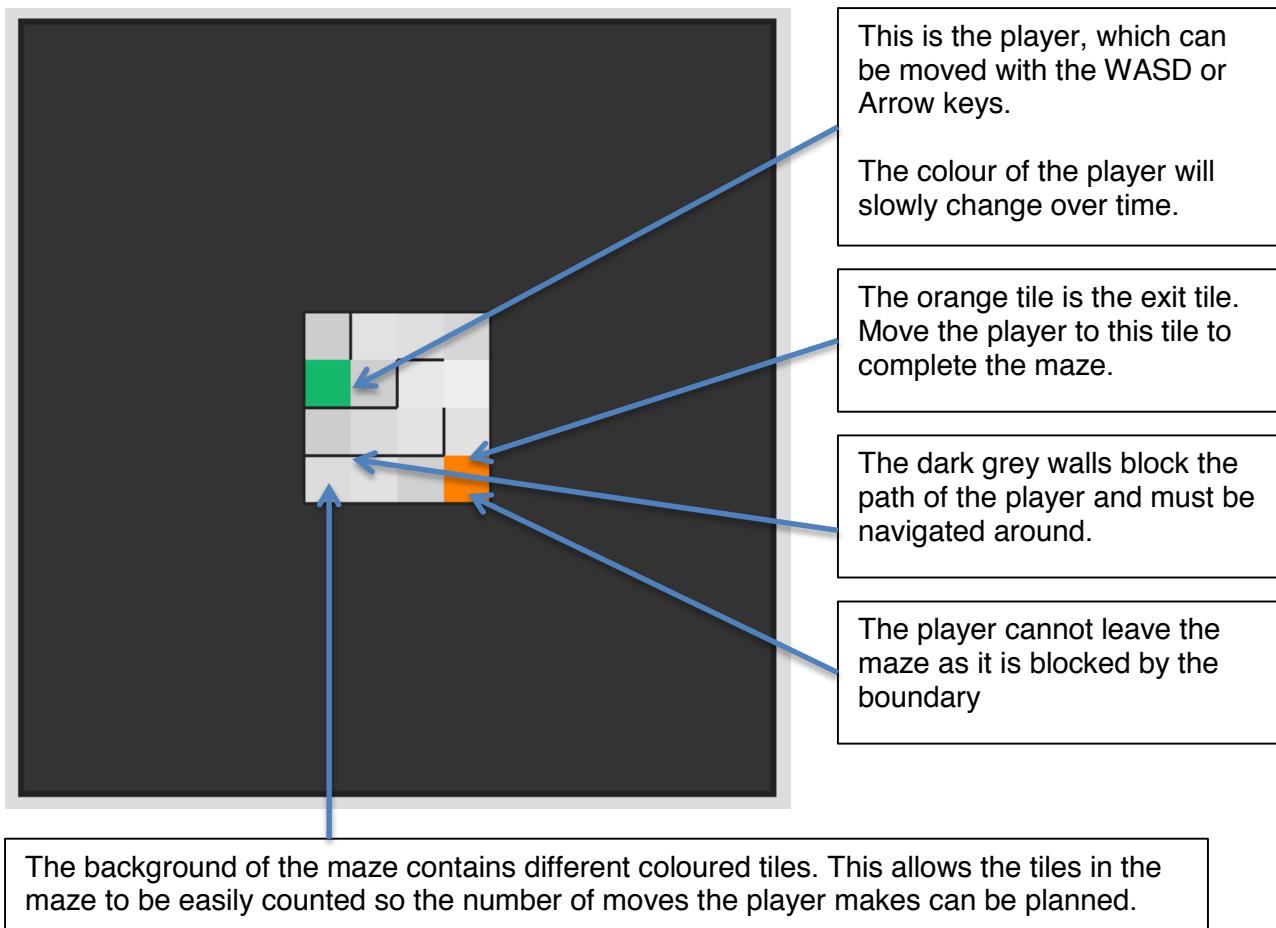
When you first visit the Mazr webpage, you'll see the main menu as shown in the image below.



By using the mouse / trackpad connected to your PC you can move the cursor over the buttons in the menu. Pressing the right button on the mouse / trackpad will press the button. The purpose of the three buttons is summarised below: Move your cursor over the New game button and press the right mouse / trackpad button. This will load a new 4 by 4 tile maze. The section that follows describes how to solve mazes.

### Solving mazes

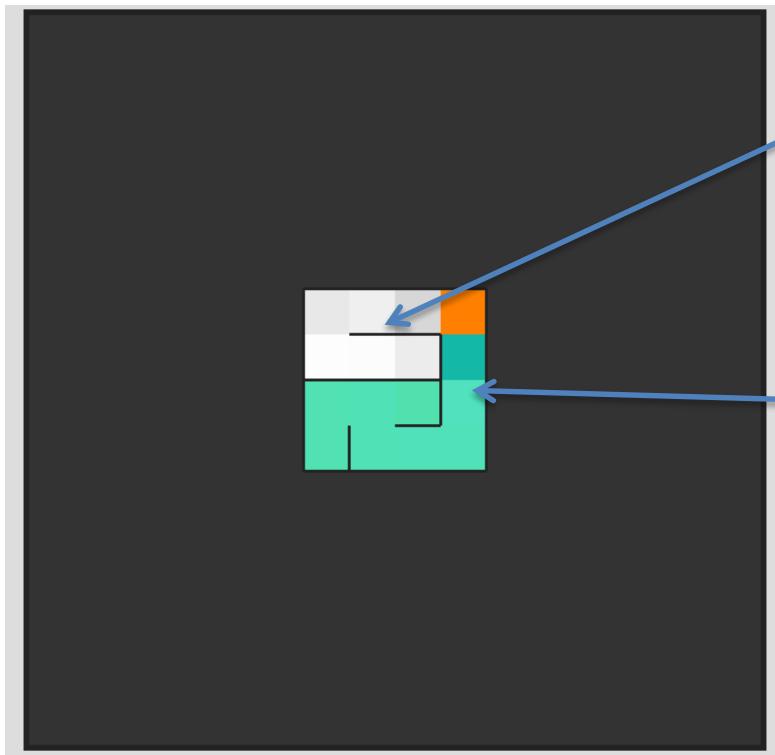
A typical first level maze, which is 4 by 4 tiles in size, is shown in the image below. All of the important game elements are labelled with descriptions.



Below is a table that summarises the purpose of each of the keys that have a function when playing the game:

Key	Description
W key / Up arrow	Moves the player up one tile, if there isn't a wall above the player or the player isn't on the top edge of the maze.
A key / Left arrow	Moves the player left one tile, if there isn't a wall left of the player or the player isn't on the left edge of the maze.
S key / Down arrow	Moves the player down one tile, if there isn't a wall below the player or the player isn't on the bottom edge of the maze.
D key / Right arrow	Moves the player right one tile, if there isn't a wall right of the player or the player isn't on the right edge of the maze.
Escape key	Toggles the pause menu. When solving a maze and the escape key is pressed then pause menu is displayed and game is paused. When on the pause menu and the escape key is pressed then the maze is displayed and game is resumed.

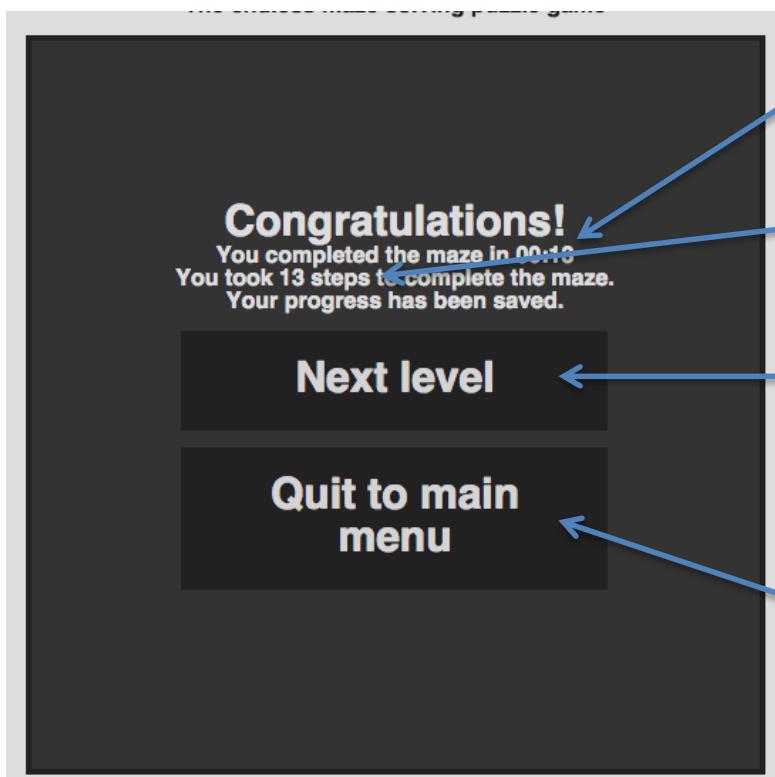
Below is an image that shows typical movement of the player along with the trail the player leaves.



Note that the trail makes it easy to see which areas of the maze are unvisited

The player leaves a coloured trail that matches its own colour. This is so it is easy to tell which parts of the maze have been visited. This becomes very helpful when the mazes get larger.

Upon moving the player up by another tile, the maze is completed and the end of level maze is displayed. This is shown on the image in the next page.



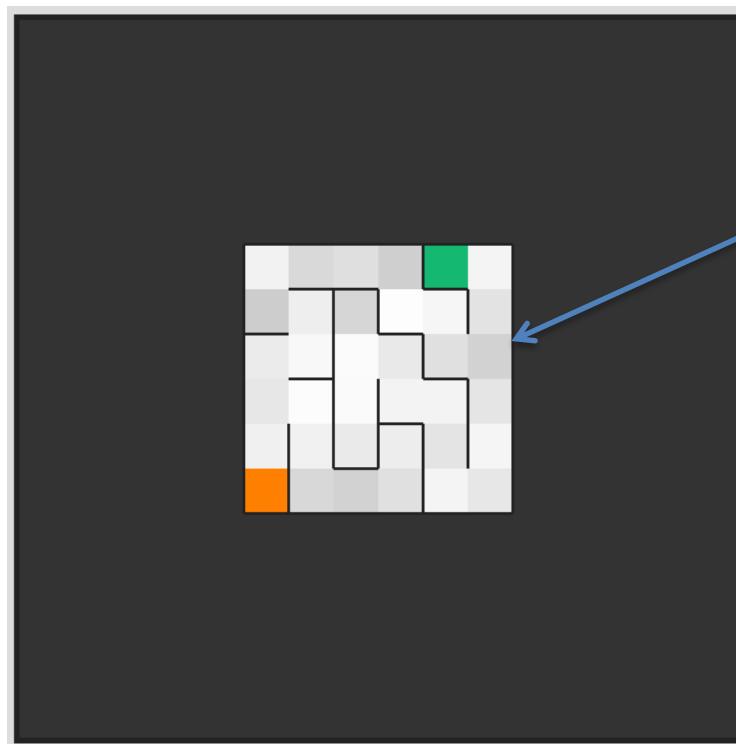
The time taken to complete the maze is displayed here.

The number of steps required to complete the maze is displayed here.

Pressing the Next level button will cause the next maze to be loaded. The new maze will be two tiles larger in each dimension than the maze just completed.

The Quit to main menu button will display the main menu.

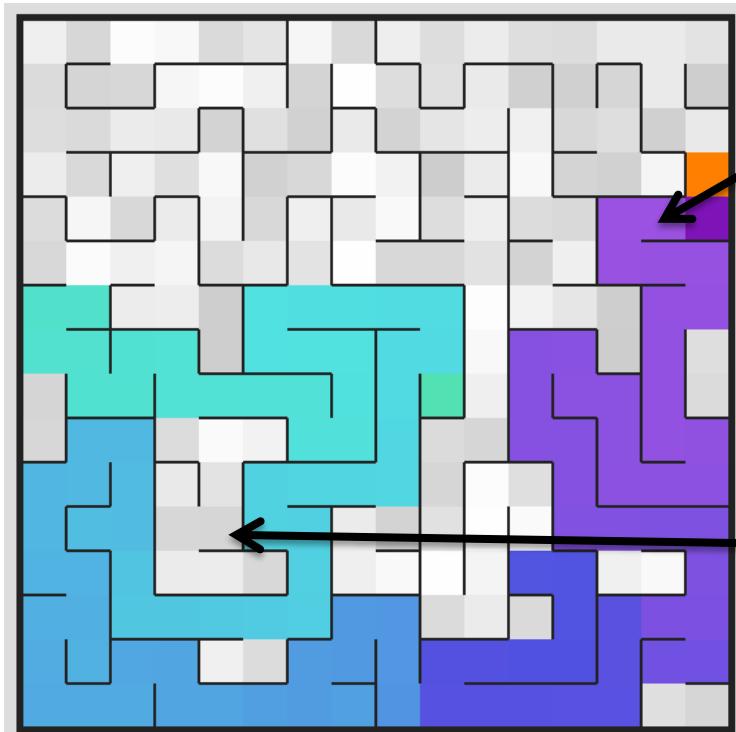
Another maze is displayed when the next level button is pressed. Mazes will get increasingly difficult as they get larger. Below is an image of a maze that was randomly generated for level 6. Note this is the largest maze that completely fits in the camera. Larger mazes are covered in the follow section.



Every time a maze is completed the next maze generated increases in size. The increase by size is two tiles in each dimension.

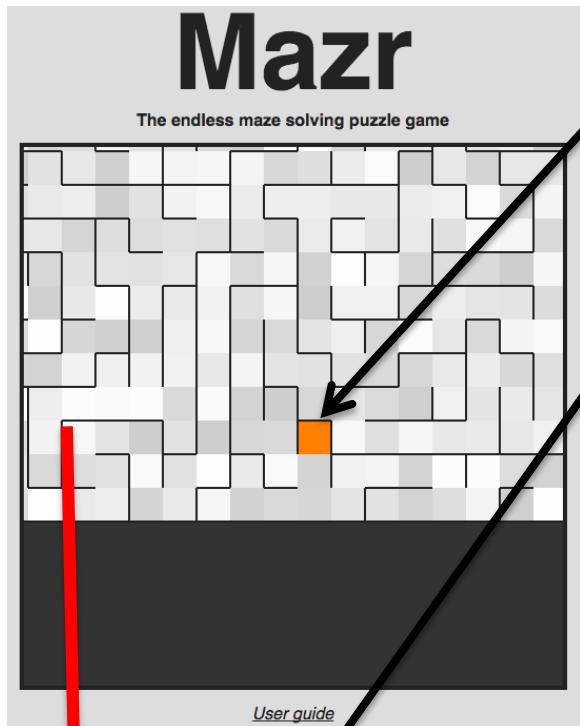
### Solving larger mazes

When the mazes become larger than the window then a camera will follow the player around. In order to make it easier to see where the exit is in a maze that does not fit in the window, the level loads with a view of the exit tile and pans over the player. This gives a general direction in which to head in order to locate the exit.



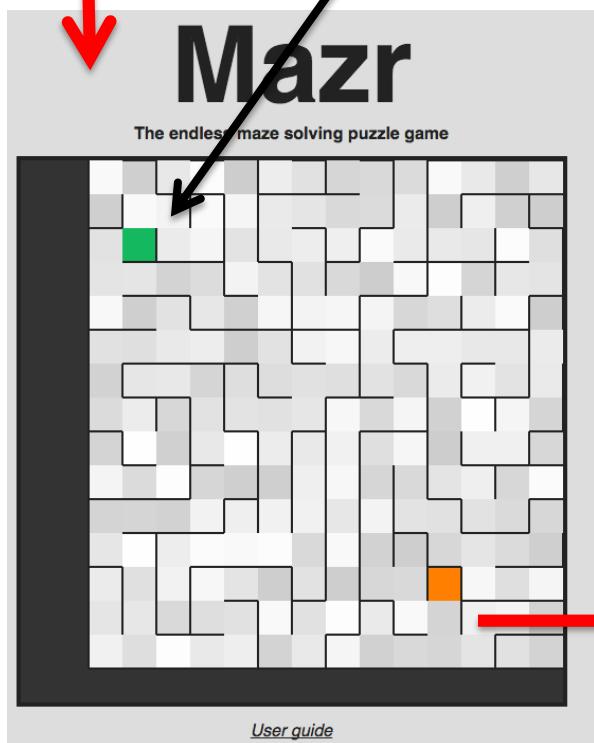
The changing colour of the player leaves a trail of different colours. This can be used to judge where has been visited in the maze.

Dead ends can quickly be identified if they are surrounded by a coloured trail.

**1.**

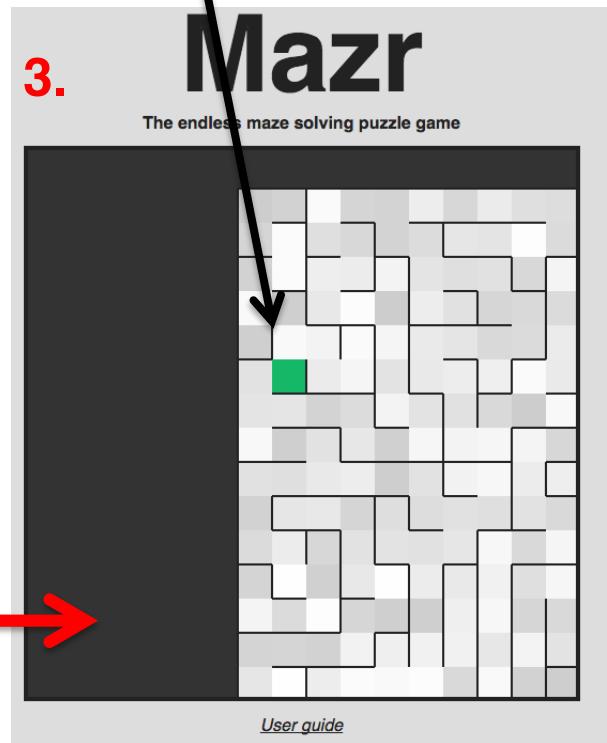
The game begins with the camera's view centred on the exit tile. Controls are disabled at this point so the player cannot be moved while it is not in view.

The camera then pans over to the player in order to give a general idea of which direction to head in order to find the exit tile.

**2.**

Finally, the camera finishes centred on the player. Controls are then enabled so the maze can now be solved.

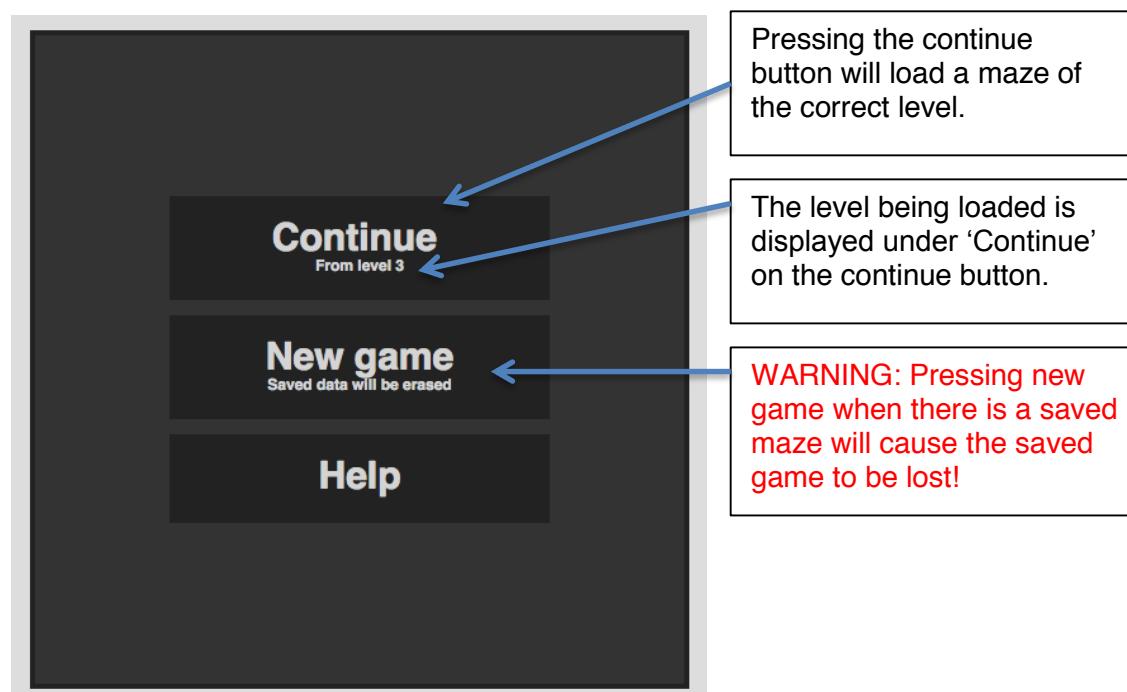
**3.**



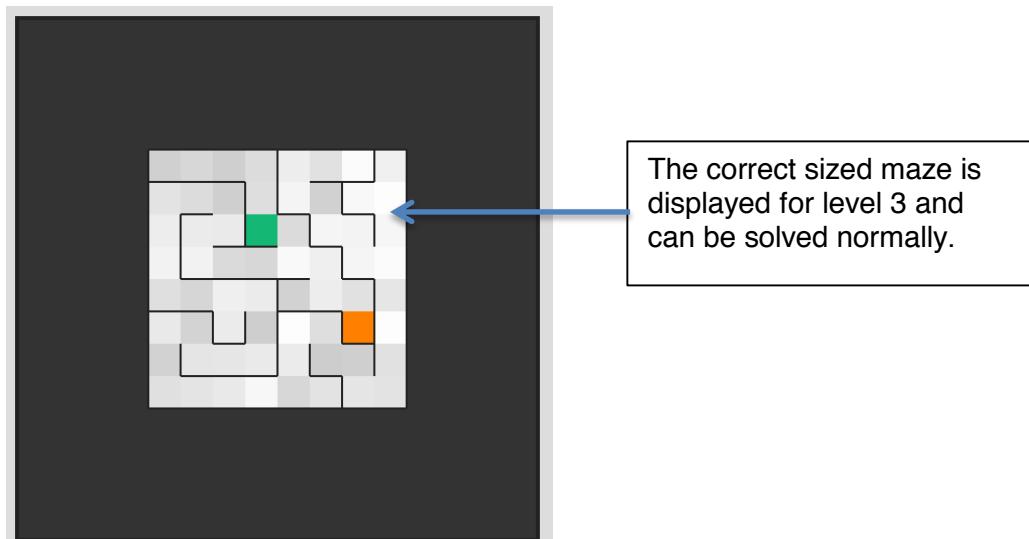
### Loading a saved maze

The game is saved whenever a maze is completed. As such quitting to the main menu, refreshing the page or closing the browser completely will not affect the saved game information. The saving is done automatically when a maze is completed so there is no need to save manually.

Mazr uses the web browser's Local Storage to store progress. Note that this is local to a computer so progress is saved only to one specific computer. All windows and tabs will update the same data so progress will be lost in one window / tab if a new game is started in another window / tab. When there a level saved, the continue button is enabled and when pressed will load a maze of the correct level and size.

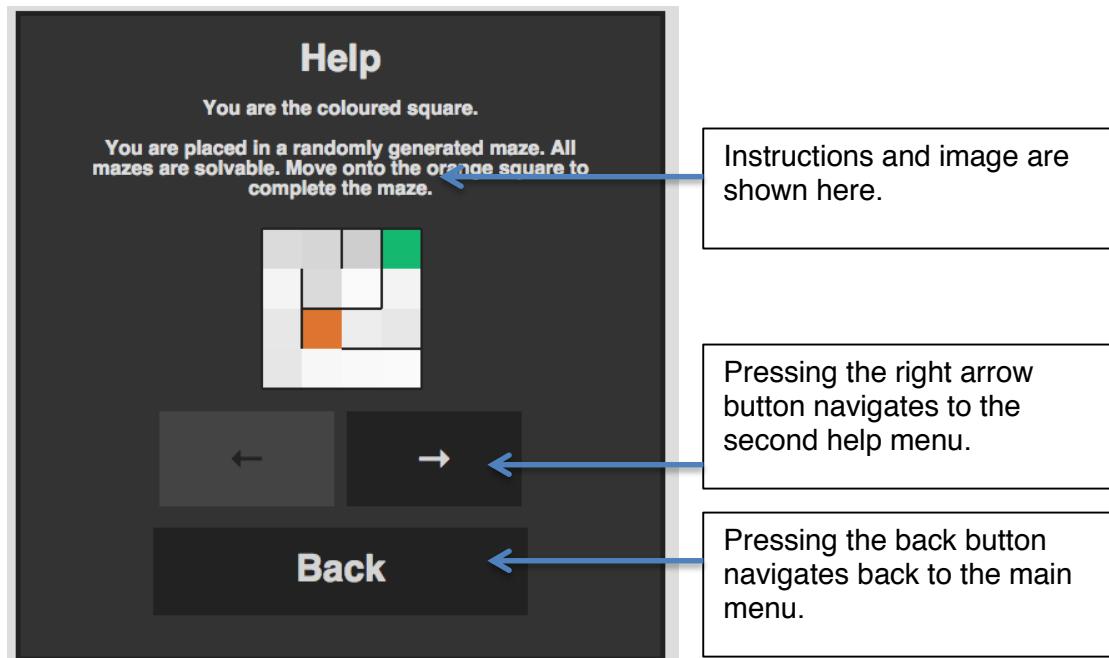


Below is an image of the maze generated and displayed when the continue button was pressed.

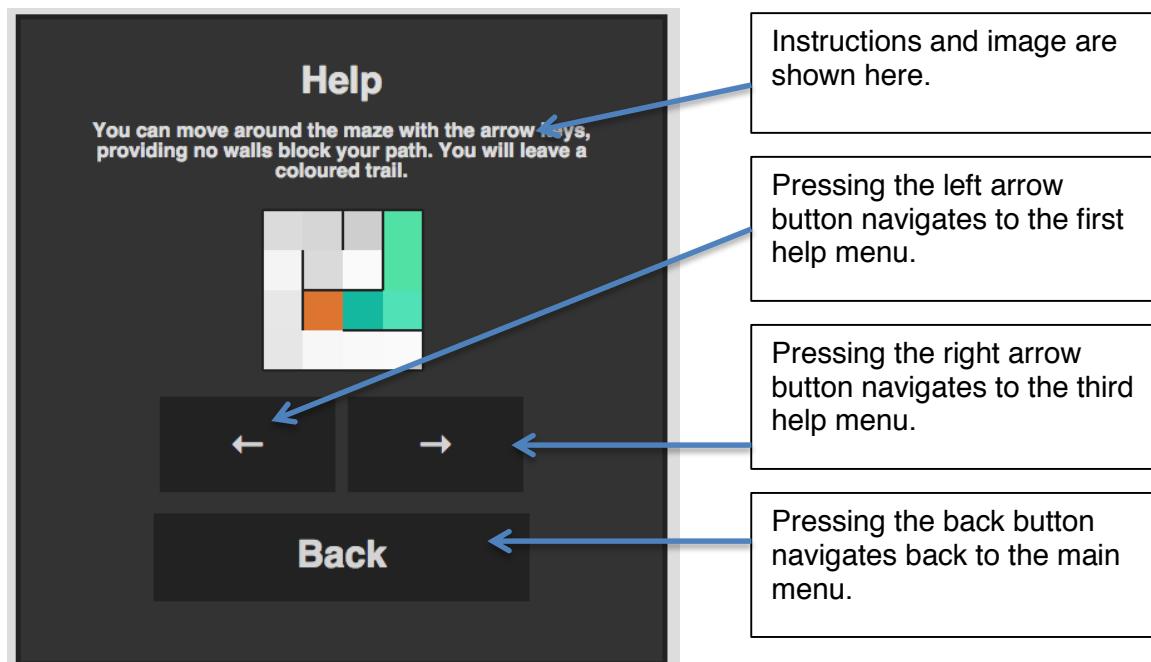


### Viewing the help menus

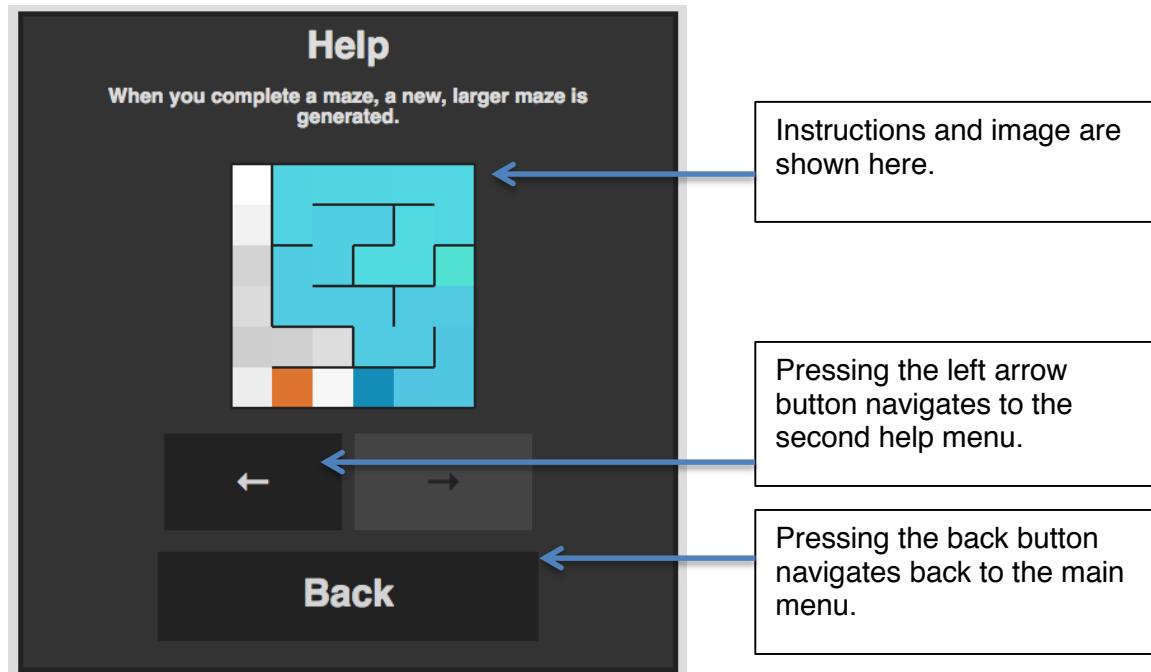
Mazr contains onscreen help that games simple instructions on how to play the game. These are included with this guide for completeness. The first image below is of the first help menu and is displayed when the Help button on the main menu is pressed (see the *starting a new maze* section for further information on the main menu).



Below is an image of the second help menu, navigated to when the right arrow button on the first help menu is pressed (as described above) or when the left arrow button on the third help menu is pressed.



Below is an image of the third help menu, navigated to when the right arrow button on the second help menu is pressed (as described above).



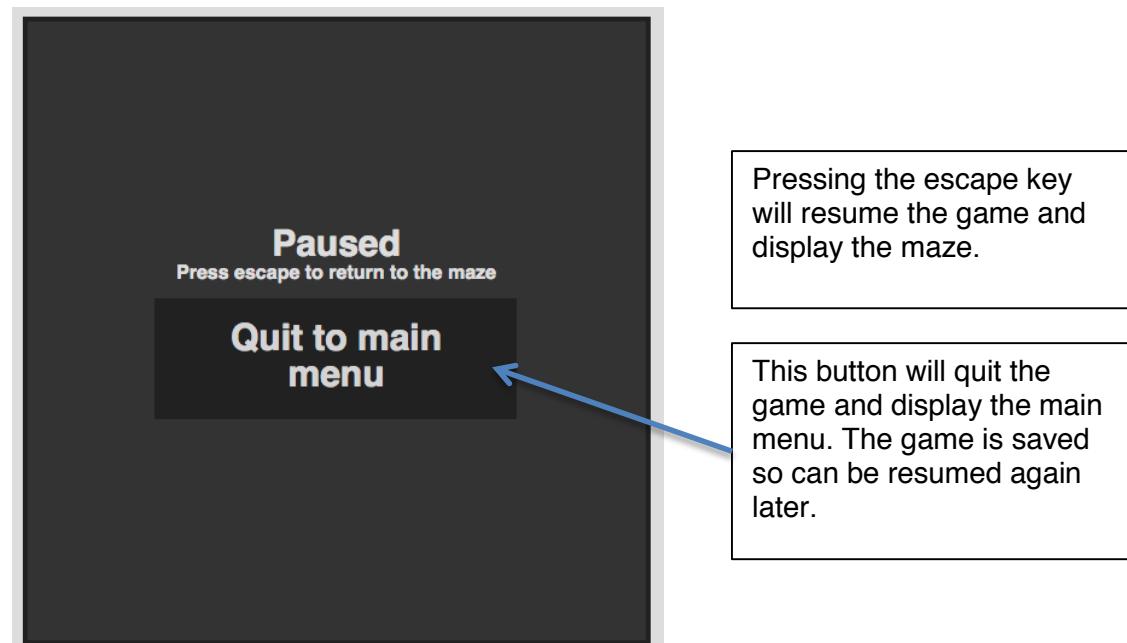
Instructions and image are shown here.

Pressing the left arrow button navigates to the second help menu.

Pressing the back button navigates back to the main menu.

### Using the Pause Menu

Pausing the game will pause the timer and display the pause menu. To game can be paused by pressing the escape key. The pause menu is shown in the image below. Note that pressing the escape key will resume the game and display the maze.



Pressing the escape key will resume the game and display the maze.

This button will quit the game and display the main menu. The game is saved so can be resumed again later.

## Troubleshooting

This section contains two tables of troubleshooting information for Mazr. Note that Mazr does not contain explicit error messages as these disrupt the immersion of the game so below are some problems that could occur when attempting to access or play the game, possible causes and their solutions.

### Accessing the game

Problem	Possible cause	Solution
Website does not load	Computer is not connected to the Internet	Connect the computer to the internet, via a physical connection or Wi-Fi.
	URL was typed incorrectly	Ensure the URL typed into the address bar is <a href="http://bit.ly/1APzjl4">http://bit.ly/1APzjl4</a>
	Mazr is currently down for maintenance / server is not functioning correctly	There is nothing that can be done about server downtime.  Regularly retry to access the webpage at the correct address.
The error message “You need to enable JavaScript in your browser in order to play the game”.	JavaScript is disabled.	Go into the browser settings of the browser being used (typically under advanced settings) and ensure JavaScript is enabled.

### Playing the game

Problem	Possible cause	Solution
The player does not move when WASD or arrow keys are not pressed.	Keyboard is not correctly connected.	Ensure the keyboard is connected, typically via USB or Bluetooth.
	Keyboard drivers are not installed.	Ensure correct drivers for keyboard are installed. Note most keyboards do not require driver installation.
The mouse / trackpad cannot be used to move the cursor.	Mouse / trackpad is not correctly connected.	Ensure the mouse / trackpad is connected, typically via USB or Bluetooth.
	Mouse / trackpad drivers are not installed.	Ensure correct drivers for mouse / trackpad are installed. Note most mice / trackpads do not require driver installation.
Game may run slowly with slow animations.	Computer does not have the minimum hardware requirements	Consult the hardware requirements section of this User Guide and ensure that the computer being used meets the requirements.
Game doesn't save progress.	Web browser setting enabled that blocks website from saving data	Check the settings of the web browser being used to ensure that sites aren't blocked from

		setting local data.
Saved game data has been lost	Playing the game in two separate windows / tabs at the same time	Only most recent game data is saved. Therefore playing in two separate windows / tabs will cause inconsistencies. Avoid by only playing game in one browser window / tab.
	Game data in Local Storage has been deleted	This could happen through a variety of ways. For example, a browser reinstall could have deleted the data. Could have also been manually deleted. There is no way to resolve this once data has been lost.

## Glossary

This section contains terms that have definitions that may not be immediately obvious. They are included in the tables of this glossary for clarification.

### Game Definitions

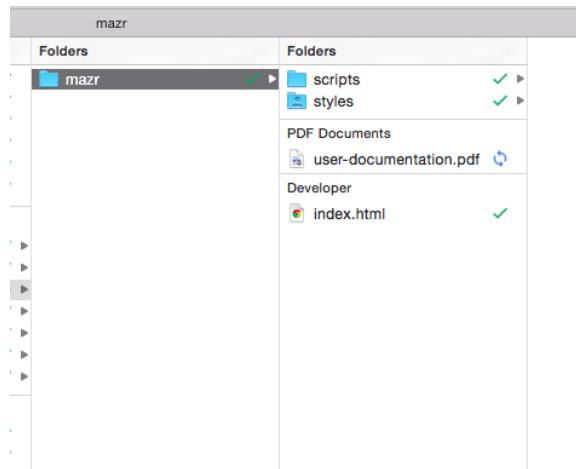
Term	Definition
Maze	A puzzle where the objective is to reach a point in a set of tiles where the path is blocked by a series of walls. The objective is to find the player a route through the path of the walls and reach the exit tile.
Player	The tile that is controlled by the user. Must be moved to the exit tile to complete a maze.
Exit Tile	The tile that is the goal of a maze. Reaching it with the player causes the maze to be completed.

### Technical definitions

Term	Definition
Operating System	Software that controls the computer hardware. Allows other programs to run on a computer.
Web browser	Software that allows for the viewing of web pages on the Internet.
JavaScript	A scripting language popular on the Internet. Must be enabled to run Mazr.
RAM	Random Access Memory. Stores programs and data on the computer.
CPU	Central Processing Unit. Processes all of the instructions on a computer.
MB (Megabyte)	Measure of information on a computer. Used to measure amount that will be required by Mazr.
Local Storage	An area where data can be saved by a website. Local to the computer. This is how the maze size and level is saved.

**Updating the user documentation**

The user-documentation.pdf file, that is currently a temporary file, can now be replaced with the finished User Documentation that is presented prior. This involves a simple switching of files, as shown in the image below.



I opened the webpage in a web browser and clicked the User guide link at the bottom of the page. The finished user documentation was displayed, which is a desirable outcome.

A screenshot of a web browser window displaying the 'User documentation' page for 'Mazr'. The title is 'Mazr User documentation'. The page includes a 'Table of Contents' with the following entries:

<b>Table of Contents</b>
Introduction.....2
System Requirements.....2
Software Requirements.....2
Hardware Requirements.....2
Accessing the game.....3
Instructions on how to play MAZR .....
Starting a new game .....
Solving mazes .....
Solving larger mazes.....7
Loading a saved maze.....9
Viewing the help menus .....
Using the pause menu.....11
Errors and troubleshooting .....
Accessing the game.....12
Playing the game.....12
Glossary.....13

The browser interface shows the URL 'file:///Users/chrisjones/Google%20Drive/SCHOOL/A2%20COMPUTING/F454/software%20development/mazr/user-documentation.pdf' in the address bar. The bottom of the browser shows various icons and a status bar indicating '2015-... 7.31.26 2015-... 7.28.55 2015-... 7.28.48'.

# Evaluation

## Degree of success in meeting the original objectives

Now that the system has been completed an evaluation of its success in targeting the problem area must be considered.

Below is a table that contains each of the final design objectives presented in the *Nature of the solution* section. These design objectives will each individually be assessed for their success along with associated evidence located elsewhere in the report that supports the degree of success that was achieved.

Objective Reference	Design Objective	Degree of success in meeting objective	Evidence of meeting objective
<b>Input Objectives</b>			
1.a.i	When the <b>new game button</b> then a new maze must be generated and displayed.	This objective was successful as a new maze is generated and displayed when the new game button is pressed.	See Test No 1 in the <i>Testing</i> section
1.a.ii	When the <b>continue button</b> is <b>not disabled</b> and pressed then saved data must be loaded from Local Storage and used to generate a maze of the saved numerical level and with the dimensions of the maze the player saved on.	This objective was successful as saved data is loaded from Local Storage and used to generate a maze of the numerical level with the dimensions of the maze the player saved on.	See Test No 2, 3 and 4 in the <i>Testing</i> section.
1.a.iii	Pressing the <b>help button</b> on the main menu will navigate the user to the first help menu.	This objective was successful as pressing the help button on the main menu displays the first help menu.	See Test No 5 in the <i>Testing</i> section.
1.a.iv	Pressing the <b>back button</b> on any of the three help menus will navigate the user back to the main menu.	This objective was successful, as pressing the back button on any of the three help menus will display the main menu.	See Test No 6 in the <i>Testing</i> section.
1.a.v	Pressing the <b>right arrow button</b> on any of the three help menus, where it is <b>not disabled</b> , will navigate the user to the next help menu. <i>For example, pressing the right arrow button on the first help menu will navigate the user to the second help.</i>	This objective was successful, as the next help menu is displayed when the right arrow button is pressed on any of the three help menus – where the right arrow button is not disabled.	See Test No 7, 8, and 9 in the <i>Testing</i> section.
1.a.vi	Pressing the <b>left arrow button</b> on any of the three help menus, when it is <b>not disabled</b> , will navigate the user to the previous help page. <i>For example, pressing the left arrow button on the second help menu will navigate the user to the first help menu.</i>	This objective was successful, as the previous help menu is displayed when the left arrow button is pressed on any of the three help menus – where the left arrow button is not disabled.	See Test No 10, 11 and 12 in the <i>Testing</i> section.
1.a.vii	Pressing the <b>quit to main menu button</b> on the pause menu or on the end of level menu will navigate the	This objective was successful, as pressing the quit to main menu buttons on either the	See Test No 13 and 14 in the <i>Testing</i>

	user to the main menu.	pause menu or the end of level menu displays the main menu.	section.
1.a.viii	Pressing the <b>next level button</b> on the end of level menu generates and displays a new maze larger by 2 tiles in the horizontal and vertical directions and with an incremented level value.	This objective was successful as pressing the next level button on the end of level menu generates and displays a maze larger by 2 tiles in the horizontal and vertical directions, along with an incremented level value.	See Test No 15 in the <i>Testing</i> section.
1.b.i	Pressing the ' <b>W</b> ' <b>key</b> will cause the player to move up by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the W key moved a player up by one tile in the maze as long as a wall was not blocking the path.	See Test No 16 in the <i>Testing</i> section.
1.b.ii	Pressing the <b>up arrow key</b> will cause the player to move up by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the Up arrow key moved a player up by one tile in the maze as long as a wall was not blocking the path.	See Test No 17 in the <i>Testing</i> section.
1.b.iii	Pressing the ' <b>A</b> ' <b>key</b> will cause the player to move left by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the A key moved a player left by one tile in the maze as long as a wall was not blocking the path.	See Test No 18 in the <i>Testing</i> section.
1.b.iv	Pressing the <b>left arrow key</b> will cause the player to move left by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the Left arrow key moved a player left by one tile in the maze as long as a wall was not blocking the path.	See Test No 19 in the <i>Testing</i> section.
1.b.v	Pressing the ' <b>S</b> ' <b>key</b> will cause the player to move down by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the S key moved a player down by one tile in the maze as long as a wall was not blocking the path.	See Test No 20 in the <i>Testing</i> section.
1.b.vi	Pressing the <b>down arrow key</b> will cause the player to move down by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the Down arrow key moved a player down by one tile in the maze as long as a wall was not blocking the path.	See Test No 21 in the <i>Testing</i> section.
1.b.vii	Pressing the ' <b>D</b> ' <b>key</b> will cause the player to move right by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the D key moved a player right by one tile in the maze as long as a wall was not blocking the path.	See Test No 22 in the <i>Testing</i> section.
1.b.viii	Pressing the <b>right arrow key</b> will cause the player to move right by one tile in the maze, unless a wall blocks them.	This objective was successful, as pressing the Right arrow key moved a player right by one tile in the maze as long as a wall was not blocking the path.	See Test No 23 in the <i>Testing</i> section.
1.b.ix	Pressing the <b>escape key</b> while a maze is displayed will cause the game to be paused and the pause menu to be displayed. If the pause	This objective was successful as pressing the escape key toggles the pause menu and the maze displaying. Pressing	See Test No 24, 25, 26 and 27 (retest) in the <i>Testing</i>

	menu is currently being displayed then pressing the escape key on the keyboard will cause the user to return to the maze. The escape key does nothing when on any other menu.	the escape key while a maze is displaying will display the pause menu. Pressing the escape key on the pause menu will resume and display the maze.	section.
Processing Objectives			
2.a.i	The maze should be generated procedurally with a randomised depth first search algorithm. All mazes must have a solution.	This objective was successful as the mazes were generated procedurally with a randomised depth first search algorithm and all mazes have a solution.	See Test No 28, 29, 30 and 31 in the <i>Testing</i> section.
2.a.ii	The first level maze should be 4 by 4 tiles in size.	This objective was successful as the first maze generated when the new game button is pressed is 4 by 4 tiles.	See Test No 32 in the <i>Testing</i> section.
2.a.iii	The player and exit zone are randomly placed in the maze at least a third of the size of the maze apart.	This objective was successful as the player and exit tile are placed randomly in the maze at least a third of the size of the maze apart.	See Test No 33 in the <i>Testing</i> section.
2.a.iv	If the maze does not fit in the canvas (512 by 512 pixels) then the camera should be enabled and follow the player around the maze	This objective was successful as the camera is enabled and follows the player around the maze when a maze cannot fit in the 512 by 512 pixel canvas.	See Test No 34 in the <i>Testing</i> section.
2.a.v	If the camera is enabled, when the new maze loads and displays for the first time, the camera should be centred on the exit zone and slowly pan back towards the centre of the player. This is so that the player gets a view of the maze's exit before they can begin solving it.	This objective was successful as the camera begins centred on the exit zone when the maze first loads and then pan towards the centre of the player.	See Test No 35 in the <i>Testing</i> section.
2.b	Each subsequent maze should be 2 tiles larger than the previous maze in the horizontal and vertical directions. <i>For example, if the 4 by 4 maze is completed then the next maze displayed should be 6 by 6 tiles.</i>	This objective was successful as subsequent mazes grow by two tiles in each dimension with respect to the maze that was last solved.	See Test No 36 in the <i>Testing</i> section.
2.c	When there is no saved game data in Local Storage the continue button should be disabled.	This objective was successful as the continue button is disabled when there is no saved game data in Local Storage.	See Test No 37, 38 and 39 in the <i>Testing</i> section.
2.d.i	The player should not move if a movement command is entered and a wall blocks the direction in which the player is attempting to move.	This test was successful as the player cannot move in a direction that is obstructed by a wall.	See Test No 40, 41, 42, 43, 44, 45, 46 and 47 in the <i>Testing</i> section.
2.d.ii	The player should not move if a movement command is entered and the edge of the maze blocks the direction in which the player is	This test was successful, as the player cannot move in a direction that is obstructed by the edge of the maze.	See Test No 48, 49, 50, 51, 52, 53, 54 and 55 in the <i>Testing</i> section.

	attempting to move.		section.
2.d.iii	The player should not move if the player is disabled. The player is disabled when menus are being displayed and when the camera is panning from the exit tile to the player (this occurs on mazes that cannot fill the canvas).	This test was successful, as the player cannot move when disabled. The player is disabled when all menus are displayed and when the camera is panning from the exit tile to the player.	See Test 56, 57 and 58 in the <i>Testing</i> section.
2.d.iv	Mark the maze as completed and load the end of level menu if the position of the player and the exit tile is the same.	This test was successful as a maze is completed (and hence the end of level menu is displayed) when the position of the player and exit tile are the same.	See Test No 28, 29, 30 and 31 in the <i>Testing</i> section.
2.d.v	As the player moves over a tile, it should colour the tile with a modified colour of the player – maintaining the same hue but with a 20% increase in lightness and 10% decrease in saturation.	This test was successful as the player leaves a coloured trail that has a 20% increase in lightness and 10% decrease in saturation of the player's colour.	See Test No 59 in the <i>Testing</i> section.
2.d.vi	The colour of the player tile should fade through a spectrum of hues slowly over time. The player should fade from hue values 145 to 300 in the HSL colour range.	This test was successful as the player's colour fades through a spectrum of hues from hue value 145 to hue value 300 in the HSL colour range.	See Test No 60 in the <i>Testing</i> section.
Output Objectives			
3.a	The game will be embedded in a single web page.	This test was successful as the game is embedded in a single web page.	See Test No 61 in the <i>Testing</i> section.
3.b.i	A header with the name of the game "Mazr" should be displayed.	This test was successful as a header with the content "Mazr" is displayed.	See Test No 62 in the <i>Testing</i> section.
3.b.ii	A subheading with the text "The endless maze solving puzzle game" should be displayed.	This test was successful as a header with the content "The endless maze solving puzzle game" should be displayed.	See Test No 62 in the <i>Testing</i> section.
3.b.iii	The game should be embedded in a HTML canvas element.	This test was successful as the game is embedded in a HTML canvas element.	See Test No 62 in the <i>Testing</i> section.
3.b.iv	A footer that contains a link to the user documentation, which will be provided as a PDF document, should be displayed.	This test was successful as a link to the user documentation is included in the footer. Note that the temporary document was replaced with the correct documentation PDF.  (See user documentation section for evidence of documentation)	See Test No 62 in the <i>Testing</i> section.  (See user documentation section for evidence of documentation)
3.c	If JavaScript is disabled in the user's browser then no menus or the canvas should be displayed and an error message with the text "You need to enable JavaScript in your browser in order to play the game" should be displayed.	This test was successful as disabling the JavaScript in the web browser causes an error message "You need to enable JavaScript in your browser in order to play the game" to be displayed and none of the	See Test No 63 in the <i>Testing</i> section.

		menus are displayed.	
3.d	Test that the game will be composed of a set of menus and each will be contained in div HTML elements and all appear on the same page although hidden / unhidden to simulate multiple pages being navigated.	This test was successful as the game are composed of a set of DIV HTML element menus and only a single menu is displayed at a given time.	See Test No 64 in the <i>Testing</i> section.
3.e.i	[The main menu contains...] A continue button, to resume a level saved in local storage. This is disabled if no game is stored in local storage.	This test was successful as the main menu contains a continue button that is disabled if no game data is saved in Local Storage.	See Test No 65 in the <i>Testing</i> section.
3.e.ii	[The main menu contains...] A new game button that starts a new game.	This test was successful as the main menu contains a new game button.	See Test No 65 in the <i>Testing</i> section.
3.e.iii	[The main menu contains...] A help button to go to the first help menu.	This test was successful at the main menu contains a help button.	See Test No 65 in the <i>Testing</i> section.
3.f.i	[All the help menus contain...] A right arrow button, which can be used to navigate to the following help page. If there is no following help page then this button will be disabled.	This test was successful as all the help menus contain a right arrow button, although this is disabled on the third help menu as there are no further help menus.	See Test No 66, 67 and 68 in the <i>Testing Section</i> .
3.f.ii	[All the help menus contain...] A left arrow button, which can be used to navigate to the previous help page. If there is no previous help page then this button will be disabled.	This test was successful as all the help menus contain a left arrow button, although this is disabled on the first help menu as there are no previous help menus.	See Test No 66, 67 and 68 in the <i>Testing Section</i> .
3.f.iii	[All the help menus contain...] A back button to return to the main menu from any of the help menus.	This test was successful as all the help menus contain a back button.	See Test No 66, 67 and 68 in the <i>Testing Section</i> .
3.g.i	[The first help menu contains...] Onscreen help that explains the format of the game and the goal of the game.	This test was successful as the first help menu contains onscreen help that explains the format of the game and the goal of the game.	See Test No 66 in the <i>Testing Section</i> .
3.g.ii	[The first help menu contains...] An image of a small maze with a player and exit tile.	This test was successful as the first help menu contains an image of a small maze with a player and exit tile	See Test No 66 in the <i>Testing Section</i> .
3.h.i	[The second help menu contains...] Onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.	This test was successful as the second help menu contains onscreen help that explains how the game controls function, how walls can block the path and the coloured trail.	See Test No 67 in the <i>Testing Section</i> .
3.h.ii	[The second help menu contains...] Image of a small maze where the player is leaving a coloured trail moving to the exit tile.	This test was successful as the second help menu contains image of a small maze where the player is leaving a coloured	See Test No 67 in the <i>Testing Section</i> .

		trail moving to the exit tile.	
3.i.i	[The third help menu contains...] Onscreen help that explains how completing a maze subsequently generates a larger maze for the player to solve.	This test was successful as the third help menu contains onscreen help that explains how completing a maze subsequently generates a larger maze for the player to solve.	See Test No 68 in the Testing Section.
3.i.ii	[The third help menu contains...] Image of a larger maze.	This test was successful as the third help menu contains image of a larger maze.	See Test No 68 in the Testing Section.
3.j.i	[The end of level menu contains...] Text congratulating the user on their completion of the maze.	This test was successful as the end of level menu contains congratulatory text.	See Test No 69 in the Testing Section.
3.j.ii	[The end of level menu contains...] Text that contains the time taken for the user to complete the maze.	This test was successful as the end of level menu displays the time taken to complete the maze.	See Test No 69 in the Testing Section.
3.j.iii	[The end of level menu contains...] Text that contains the number of steps taken for the user to complete the maze.	This test was successful as the end of level menu displays the number of steps taken by the player to complete the maze.	See Test No 69 in the Testing Section.
3.j.iv	[The end of level menu contains...] The next level button, which allows the user to proceed to a newly generated maze.	This test was successful as the end of level menu contains a next level button.	See Test No 69 in the Testing Section.
3.j.v	[The end of level menu contains...] The quit to main menu button, a button that allows the user to quit to the main menu.	This test was successful as the end of level menu contains a quit to main menu button.	See Test No 69 in the Testing Section.
3.k.i	[The pause menu contains...] The quit to main menu button, a button that allows the user to quit to the main menu.	This test was successful as the pause menu contains a quit to main menu button.	See Test No 70 in the Testing Section.
3.k.ii	[The pause menu contains...] Small amount of text expressing that pressing the escape key will return the user to the maze.	This test was successful as the pause menu contains text that explains pressing the escape key on the pause menu resumes the maze.	See Test No 70 in the Testing Section.
3.l.i	All of the game elements (such as tiles, the maze, player, exit tile, etc.) will be drawn to the screen with a HTML 5 canvas element.	This test was successful as the game elements are rendered in a HTML 5 canvas element.	See Test No 62 in the Testing section.
3.l.ii	The canvas should be encapsulated within a 512 by 512 pixel bordered container.	This test was successful as the canvas is contained within a pixel-bordered rectangle.	See Test No 71 in the Testing Section.
3.l.iii	A grid of tiles with up to four walls on each side that represents the maze must be drawn to the canvas, along with a player and exit tile.	This test was successful as the maze is rendered as a grid of tiles with up to four walls along with a player and an exit tile.	See Test No 72 in the Testing Section.
3.l.iv	The tiles in the game should be 32 by 32 pixels.	This test was successful as tiles in the game are 32 by 32 pixels.	See Test No 73 in the Testing Section.
3.l.v	The player tile should be 32 by 32	This test was successful as the	See Test No 74

	pixels.	player in the game is 32 by 32 pixels.	in the <i>Testing Section</i> .
3.l.vi	The exit tile should be 32 by 32 pixels.	This test was successful as the exit tile in the game is 32 by 32 pixels.	See Test No 75 in the <i>Testing Section</i> .
3.l.vii	The walls of the maze should be 32 by 4 pixels.	This test was successful as the walls of the maze in the game are 32 by 4 pixels.	See Test No 76 in the <i>Testing Section</i> .
3.l.viii	The colours of tiles in the maze (excluding player and exit tiles) should be an assortment of shades of grey with an RGB colour value in the form $RGB(\alpha, \alpha, \alpha)$ where $200 \leq \alpha \leq 255$ .	This test was successful as the colours of the tiles in the maze are an assortment of shades of grey.	See Test No 77 in the <i>Testing Section</i> .
3.l.ix	Saturation and lightness of player tile should remain constant at values of 80% and 40% respectively.	This test was successful as the hue and saturation of the player remained at constant values of 80% and 40%.	See Test No 78 in the <i>Testing Section</i> .
3.l.x	The colour of the exit zone should be orange with hex value #FF8000.	This test was successful as the colour of the exit tile has the HEX value #FF8000.	See Test No 79 in the <i>Testing Section</i> .
3.l.xi	If the maze fits in the canvas (512 by 512 pixels) then the maze should be centred in the canvas and the camera disabled.	This test was successful as the camera is disabled and the maze is centred in the canvas if the maze fits in the canvas.	See Test No 80 in the <i>Testing Section</i> .
3.l.xii	If the maze doesn't fit in the dimensions of the canvas (512 by 512 pixels) then a camera should be enabled that has the player positioned at the centre and follows the centre of the player, after panning has occurred.	This test was successful as the camera is enabled and positioned to follow the centre of the player around the maze (after panning from the exit tile has occurred) when the maze doesn't fit in the canvas.	See Test No 34 in the <i>Testing Section</i> .
3.l.xiii	The hue of the player must constantly change between the values 145 to 300.	This test was successful as the hue of the player changes between values 145 and 300.	See Test 60 in the <i>Testing section</i> .
3.l.xiv	The player's trail must be displayed. Each tile visited by the player must be coloured with the same hue but a 20% increase in lightness and a 10% decrease in saturation.	This test was successful as the player leaves a trail that a hue equal to that of the player at a given time, with a 20% increase in lightness and a 10% decrease in saturation.	See Test No 59 in the <i>Testing section</i> .

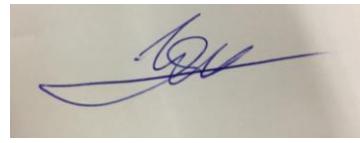
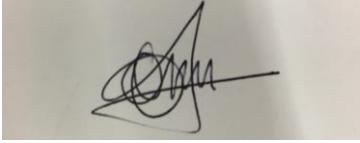
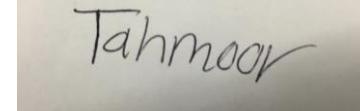
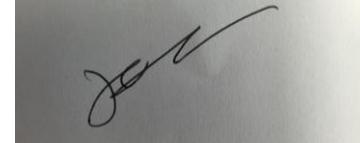
## User evaluation

The user evaluation is an extension of the feedback that was received from the end users at the Testing stage. This will provide a more qualitative review of the user's opinions on the system and is provided below. Each user and their answer is summarised in the table on the following page.

Below is a summary of the column headings in the table:

- **Signature of user** – This is to certify that the user agrees that these were their opinions that have been reproduced at this stage.
- **Responses to the following questions** – Under this column is a series of columns that contain textual responses in the form of quotes as answers to each of the questions. I will evaluate any issues after the table has been presented.

**User evaluation table**

Signature of user	Responses to the following questions:				
	Were the menus clear?	Was the onscreen help sufficient?	Were the controls easy to use?	Were you happy with the finished game?	Do you have any criticism of the game?
	'The menus were very easy to use'	'Yes.'	'Very easy.'	'I couldn't stop playing it so I'd say I'm very happy.'	'No criticism.'
	'Very clear.'	'Yes.'	'Yes.'	'Yes.'	'I have no criticism.'
	'I had no problems with the menus.'	'There was only a small amount of help but it didn't matter because the game is very easy to pick up and play.'	'Really liked that I could use WASD or the arrows. So that's a yes.'	'I really enjoyed the game. It's great fun.'	'I had no problems with the game.'
	'The menus were excellent.'	'Yes.'	'Yes.'	'Yes. It's really good.'	'I have nothing bad to say about the game.'
	'Menus were great. No issues at all.'	'Yes.'	'Yes.'	'For the most part, I was happy. I only have a small issue.'	'Could only save the game on one computer. If I wanted to continue the game at home or at school I'd have to start a new game.'

# Evaluation of user's response to the system

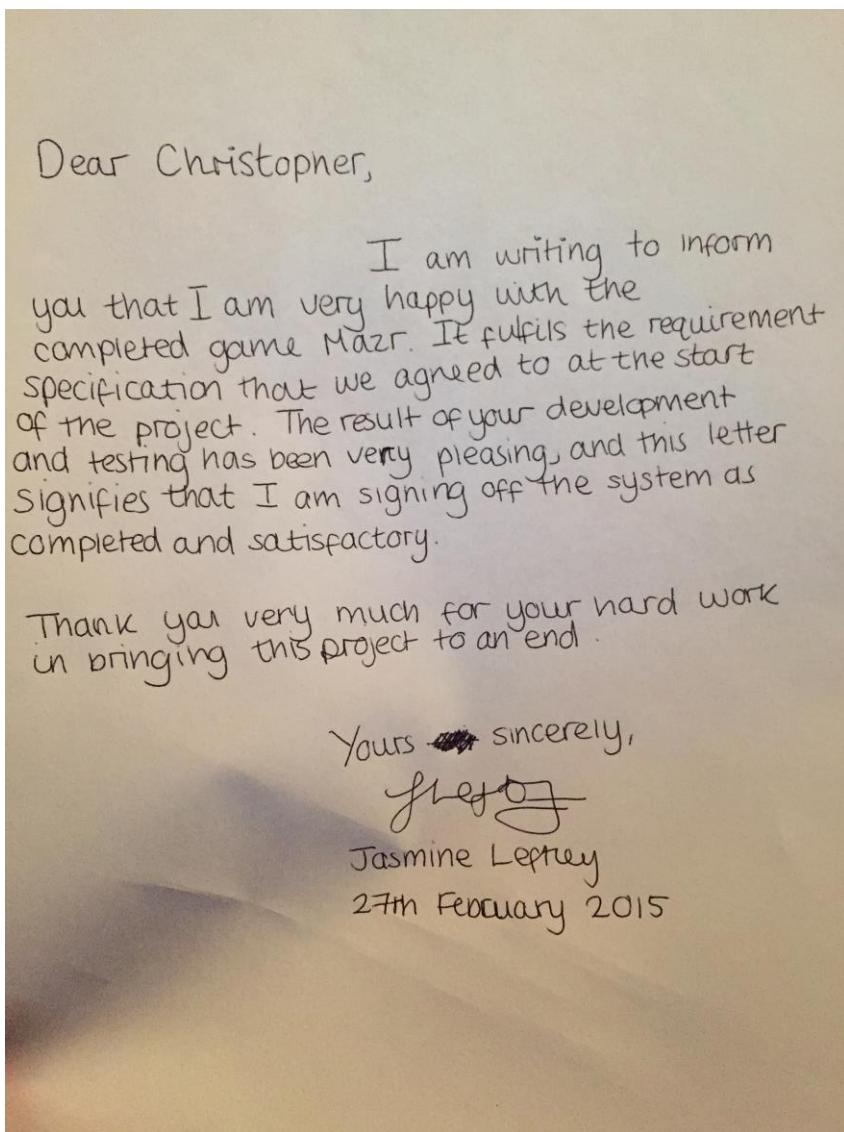
## Introduction

The user evaluation above, along with the end user testing stage in the Testing section, shows that most of the comments made by the users are positive. This is a very good response for the most part.

One of the responses from the users was that there 'could only save the game on one computer. If I wanted to continue the game at home or at school I'd have to start a new game'. This is a feature that is beyond the scope of this project at the current time and will be discussed in the desirable extensions section.

## Letter from client

The client also produced this letter that represents their sign off on the system and any final remarks they have. They have signed and dated this letter and it is shown in the image below.



# Desirable Extensions

## Introduction

This section contains some reflective information on the good and bad points of the final system along with an assessment of the system's limitations. Finally there is an examination of extensions that could be made and how the any extensions could be carried out.

## Good points of the system

- The completed system, after rigorous testing, successfully met all of the Design objectives in the Nature of the solution section produced with feedback from the client. This was an excellent result because the Design objectives were a lengthy list of various strict objectives. Any initially unsuccessful objective tests were rectified and repeated to ensure they were met.
- The game was a product of close collaboration between the client and myself and the client was at the end very happy with the system I had produced. This is evident as they signed off on the testing and the evaluation stage, as they believed the game was a completed product. This was a pleasing result.
- Large proportion of end users enjoyed the game as they had very few negative comments. Any possible modifications were made in an attempt to respond to end-user testing. End user feedback also noted positive comments from the end users.
- The game worked with no detectable errors or bugs. Of course it is unrealistic to say that none will ever be detected although at this point it is a good indicator that the system has very few errors or bugs that will be detected.
- The game is an enjoyable experience as noted in the user feedback.
- The maze-generating algorithm works excellently for the game. In all of the testing and user involvement there is no noticeable repetition in maze patterns. This worked vastly better than expected. This is evident in that there are no repeated mazes contained in any of the images in the report that display a maze. This resulted from the research and then implementation of the randomised depth first search algorithm.
- The game's graphical styles and menus closely resembled the designs and screen layouts in the Nature of The Solution. This is evident by comparing images of the completed game with the screen layouts and the comments made by the client.
- The game was a good level of difficulty and learning curve as noted in user feedback.
- The user feedback noted excellent onscreen support in their comments about the help menus.
- The game's controls were noted as simple to use by the users in the user feedback and the user evaluation

## Bad points of the system

- There is no storing of user's steps taken or time taken to solve a given maze and no method to review these scores on previous mazes. Although I'm disappointed this wasn't included, it was overlooked during discussions about features that were very important in the game and more crucial to be included in the time format given.

## Limitations of the system

The game is limited primarily by being a client side application only – all scripts and dynamic elements are executed on the client's computer after having been served the files. This provides a limitation as it means all code is restricted to a single computer with no ability to network with other users or a centralised backend. Therefore there is no way to share high scores or to store the current level or the time taken for each maze.

There is also an issue in that levels are only stored in the Local Storage key-value system of the web browser and can easily be modified in the console to display very large levels. For the scope of the

project given in this report it is not necessary to provide any form of encryption (as the end user group is small and the game is a novelty and fun and does not contain a highly competitive element).

## Possible extensions

The game could primarily be improved with the inclusion of a backend architecture involving a programming platform that could run scripts on a server. This would provide a whole plethora of possible extension features that would complement the game. The inclusion of a database would allow users to have a 'Mazr account' and store details such as a username and the highest level they've reached. This would provide a more secure way of storing levels and a method of displaying a leader board for the highest achieving users of the game.

### How the extensions could be carried out

A backend server could be configured to run code that serves dynamically generated HTML to the users. This would involve the use of another programming language. I would use either Ruby with the Sinatra framework or JavaScript with the Node.js framework, as I am knowledgeable in both these languages.

A relational database would need to be created that contains a table that contains all of the users of the game. A hashing algorithm would have to be used to encrypt sensitive information (such as a password). Then, a related table could contain a record of the steps and time taken for each player to complete a specific maze in a 'one to many' relationship. This would allow for a scoreboard to be generated that the users could view to review their progress against other users. Text input forms would also have to be integrated into the menus for the games. This would be necessary because a login menu would be required so that the user could sign in and retrieve save data that is stored in a database on the server.