|  |
| --- |
| Staffordshire University |
| Maze Mapping and Traversal using a Micromouse |
| An attempt at designing and creating a small, cheap and mobile single-board robot that can navigate a maze efficiently and unassisted |

|  |
| --- |
| CASHMORE Jasper  12-15-2014 |

Table of Contents

[Problem Specification - The Maze 2](#_Toc409347399)

[Outline 2](#_Toc409347400)

[Maze specifications 2](#_Toc409347401)

[Suitable and unsuitable mazes 2](#_Toc409347402)

[Solution 3](#_Toc409347403)

[Goal of the implemented solution 3](#_Toc409347404)

[The Micromouse 3](#_Toc409347405)

[Building the maze 4](#_Toc409347406)

[Possible materials 4](#_Toc409347407)

[Cardboard Maze Design 4](#_Toc409347408)

[The Robot 5](#_Toc409347409)

[Requirements 5](#_Toc409347410)

[Build or Buy 5](#_Toc409347411)

[Decision Table 6](#_Toc409347412)

[Building the robot 7](#_Toc409347413)

[Pre-built Candidates 8](#_Toc409347414)

[Methodology 11](#_Toc409347415)

[Traditional Methodologies 11](#_Toc409347416)

[Waterfall 11](#_Toc409347417)

[Unified Process 11](#_Toc409347418)

[Spiral Model 12](#_Toc409347419)

[Agile Methodologies 12](#_Toc409347420)

[Extreme Programming 13](#_Toc409347421)

[Scrum 13](#_Toc409347422)

[Rolling Wave Planning 13](#_Toc409347423)

[Conclusion 13](#_Toc409347424)

[Pathfinding and Navigation 14](#_Toc409347425)

[Bibliography 19](#_Toc409347426)

# Problem Specification - The Maze

## Outline

Given a physical set of objects that form a maze-like structure, an entity needs to get from one position in the maze to another via the fastest route possible. The entity should not touch the maze’s walls, nor should it receive information about the maze’s structure from any external source, relying solely upon its own sensors and mechanisms to learn by itself.

## Maze specifications

The maze’s structure will be defined as follows:

* It should be modifiable to allow restructuring
* It must be closed, i.e. have no exit to the outside of the maze
* All the walls must be angled on 180° sides. E.g. No diagonal walls
* Every wall’s length must be rationally divisible by the same number, e.g. a wall could be 10cm, 20cm, 30cm or 40cm long but not any other value that isn’t perfectly divisible by 10
* The thickness of the path through the maze should always be the minimum length of a wall (10cm for example)
* It should have a start point and an end point that are identifiable by a sensor
* It must be possible to travel from the start point to the finish

## Suitable and unsuitable mazes

The following illustrate all possible valid and invalid maze structures. The solution to the problem need only attempt at solving valid mazes.

|  |  |
| --- | --- |
| **Valid Maze**  Start and Finish are accessible inside the maze | **Valid Maze**  Multiple routes to the finish with dead-ends |
| **Valid Maze**  Loops and many routes to finish | **Invalid Maze**  Has openings |
| **Invalid Maze**  No possible route to the finish | **Invalid Maze**  Varying thickness of the corridors |

# Solution

## Goal of the implemented solution

The ultimate goal of this project should result in a robot that is or can navigate a maze from a start point to an end point via the most efficient route possible without any aid. Stemming from this are the specific physical features required by the robot required to achieve said goal:

* Ability to move forward by a distance specified by the software
* Ability to rotate up to 180° from its current heading
* Self-powered. i.e. Own power supply - no power cables need be attached
* Avoids contact with walls

This research and analysis documented in this report will determine the specific process and general working of the Micromouse, thus defining the hardware and software requirements that will be necessary to achieve the above goals. These requirements will be drawn up at the end of this document.

## The Micromouse

Research into this project’s field revealed existing solutions to the problem in the form of single-board, minimal design robots known as “Micromice” designed for the purposes of competitions. These competitions officially began in the 1970’s and are commonly held in the UK, the US and various parts of Asia.

The aim of a Micromouse competition is to reach the centre of the maze in the fastest time. The start and finish points are known, but the maze’s structure is not made aware to the entries, however a short two minute period is available for the Micromice to explore and form virtual maps of the maze before their entry actually begins.

Factors which play a part in an entry’s chance of winning are the algorithm it uses to map the maze and its hardware such as motor, wheels and general weight. The Path finding algorithms used that will result in profitable routes are commonly known and widely available for the public, however the best algorithms are kept secret, states (Willardson, 2001).

For this project, a Micromouse will be developed to solve the maze. Research will go into analysis of hardware, path finding algorithms and compatibility between specific components.

Primarily, focus will be put on developing a solution that can solve the maze using four-way movement but a potential stretch goal will be implementing diagonal movement if there is time remaining to do so.

# Building the maze

### Possible materials

|  |  |  |
| --- | --- | --- |
| **Material** | **Advantages** | **Disadvantages** |
| LEGO® / LEGO® Duplo | Easy to setup and readjust | Expensive for the amount that would be needed |
| Cardboard | Cheap | Would require time spent to craft the maze |

## Cardboard Maze Design

The cardboard maze will be based on a large square piece of reinforced card. The size of this will vary depending on the size and movement style of the robot so this will be decided at a later stage.

This base will be raised above the surface it’s on by 1-2cm. It will then be split into a grid and slits will be cut into each section of the grid (all the black and red lines in the diagram to the right).



Maze3DThe walls will be made of thinner card that can be slotted into the base to form the pathways of the maze.

**Maze Terms**

|  |  |
| --- | --- |
| **Wall** | A wall that is built of card that the robot can detect but cannot pass through |
| **Non-wall** | A side of a tile that is not a wall |
| **Tile** | A ‘square’ of the maze that has four sides: North, East, South and West. Any of these sides can be a wall or a non-wall |
| **Dead-end** | A tile whereby three of its four sides are walls |
| **Loop** | A collection of one or more joined sides that does not attach to the maze perimeter at either end |
| **Start** | A coloured square drawn on one of the tiles denoting the beginning point of the maze |
| **Finish** | A coloured square drawn on one of the tiles denoting the end point of the maze |
| **Path** | A journey from one tile to another |
| **Route** | A path from the start to the finish |

# The Robot

## Requirements

A suitable robot must be chosen for this project, the requirements are as follows:

* Is fully programmable
* Should be relatively small
* Capability to move forwards (backwards is ideal but optional)
* Capability to turn up to 180° (360° ideal but optional)
* Can run from a reliable, affordable source of power (alkaline battery, typical AC, USB, etc.)
* Has a suitable sensor built-in or the option of attaching one to it

## Build or Buy

One of the foreseen questions is whether the robot should be built from hand using several components such as the board, some wheels, a sensor and such, or whether a pre-built one should be bought to avoid potential complications with building one.

### Decision Table

Each point can have a maximum score of 50 and a minimum of -50. The cost of these weightings are determined based on how much impact the factor has on the overall decision.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Build** | **Score** | **Buy** | **Score** |
| **Advantages** | Complete flexibility and control over the design and creation of the robot and no more | +45 | Saves time that could be spent on development | +35 |
| Only need to buy the exact components required for the project | +10 | May find a solution that is designed for the exact or similar purposes | +45 |
| Likely to be cheaper than a pre-built solution | +30 | Will be tested by previous consumers | +40 |
| A huge amount of options, ultimately deciding what the final robot will be capable of | +40 |
| **Disadvantages** | As the separate parts may not have been tested together before, problems could occur when assembling the robot | -20 | May come with unnecessary parts | -5 |
| Lots of time and research needed to find all the best components on a limited budget and time spent assembling them | -35 | Likely to be the more expensive | -30 |
| A very limited number of choices of pre-built robots | -25 |
| **Total** |  | **70** |  | **55** |

### Building the robot

Building the robot from several components is the optimal solution for meeting the requirements and having complete control over the design. The robot would be built to meet the project’s goals making it a more favourable solution. However the opportunity cost of doing this is high as more time would need to be dedicated to this process that could otherwise be spent working on the maze and navigation algorithms.

A possibility would be to seek assistance for this process from an electronic engineer.

### Pre-built Candidates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Image** | **Name** | **Price** | **Notable Specs / Features** | **Other** | **Link** |
| http://www.picaxe.com/Thumbnail.ashx?image=~/Site_Resources/Media/Site_1/bot120/BOT120.jpg&h=600&w=800&mode=Absolute&k=2c044d78a8e5c71d4623dd3169dd9da000e9ff7b1efcb317174519cb7c7c104c | PICAXE-20X2 | £48 including UK shipping cost | * Bumper module * Line follower module | * Is a barebones robot but extra components are easy to attach, although this would add extra cost and complications * BASIC only, no support for other languages or frameworks | [PICAXE](http://www.picaxe.com/Hardware/Robot-Kits/PICAXE-20X2-Microbot/) |
| http://www.picaxe.com/Thumbnail.ashx?image=~/Site_Resources/Media/Site_1/kit110/kit110.jpg&h=600&w=800&mode=Absolute&k=0e00573d1a61a7119e640e40734c0d09a9f60d8adaf07e6e1eadfc81b6f263dc | PICAXE PICone | £90 including UK shipping cost | * PICAXE-28X2 microcontroller running at 64MHz | * Designed specifically for Micromouse competitions, making it a suitable choice * BASIC only, no support for other languages or frameworks | [PICAXE](http://www.picaxe.com/Hardware/Robot-Kits/PICAXE-PICone-Micromouse/) |
| Pololu 3pi Robot Kit | Pololu 3pi | £70 (free shipping) | * C-programmable Atmel ATmega328P microcontroller * 32KB flash memory, 2 KB RAM and 1 KB EEPROM * Capable of speeds exceeding 3 feet per second | * Designed for line-following, doesn’t have the necessary requirements for a maze with walls | [Pololu](http://www.pololu.com/product/975)  [Hobbytronics](http://www.hobbytronics.co.uk/3pi-robot) |
| http://ecx.images-amazon.com/images/I/81NIouhWlwL.jpg  https://www.bananarobotics.com/shop/image/cache/data/sku/BR/0/1/0/0/6/BR010062-DFRobot-2WD-MiniQ-Robot/DFRobot-2WD-MiniQ-Robot-top-600x600.jpg https://www.bananarobotics.com/shop/image/cache/data/sku/BR/0/1/0/0/6/BR010062-DFRobot-2WD-MiniQ-Robot/DFRobot-2WD-MiniQ-Robot-bottom-600x600.jpg | MiniQ 2WD | £60 including shipping cost from the US | * Fully programmable Atmel ATmega328P microcontroller robot brain. * Five bottom facing IR reflectance sensors for map navigation and edge detection. * Two forward facing IR transmitters for obstacle avoidance. * One forward facing IR receiver for obstacle avoidance and remote control. * Two forward facing CDS photocells for seeking light or dark. * Two motors with encoders to fully control the speed of movement. * Travels at speeds up to 31 inches per second. * A ball caster for stability. * Three user input buttons wired to a single analog port. * A buzzer for making beeps and playing tunes. * An ICSP interface for advanced programming. * Handy power and reset buttons | * Fulfils the requirements and would be easily extensible extra goals | [Banana Robotics](https://www.bananarobotics.com/shop/DFRobot-2WD-MiniQ-Robot)  [Amazon](http://www.amazon.co.uk/MiniQ-2WD-Complete-Kit/dp/B00E68HY88/ref=sr_1_cc_1?s=aps&ie=UTF8&qid=1417972564&sr=1-1-catcorr&keywords=4WD+MiniQ) |
| iRobot Roomba Vacuum | iRobot Create | £100 including shipping cost from the US | * Caster, left, and right wheel drop sensors * Left and right bumper * Wall sensor * Left, front left, front right and right cliff sensors * Omnidirectional IR receiver | * No longer available to purchase (was available at the start of the project) * Lacks much internal processing and computing power, would need this attached * Expensive | [iRobot (page removed)](http://store.irobot.com/education-research-robots/irobot-create-programmable-robot) |
| http://www.robotstorehk.com/micromouse/images/RJ_linetracer.gif | RS-Cruiser | £87 including shipping cost from China | * Size: 82 × 125 × 60 (H) mm * CPU Board: Atmega8 (8-Bit RISC MCU) * Sensor: Line sensor module- Infrared emitter & sensor × 6, Wall sensor module- Infrared emitter & sensor × 6 * Motor: MRM-GM03 Gearbox (DC Motor × 2) * Speed: 0 - 50 cm/s * Display: LED × 6 * Key: RESET × 1, User Push Button × 2 * Battery: AA × 4 (not included) * ATmega8 CPU (8Kb In-System Programmable flash memory) * Line detecting sensor module & wall detecting sensor module * Free C compiler(WinAVR) | * Very small website and company that is based in China – could be complications * Hardly any information about it online | [robotstorehk](http://www.robotstorehk.com/micromouse/RS-CRUISER.HTML) |

# Methodology

The two most popular styles of software methodology can be categorised by the terms ‘traditional’ and ‘agile’. To form the methodology for the project several development strategies from each will be researched and a methodology will be decided upon based on its suitability for this project. Should no specific methodology deem itself fully applicable for the project a ‘hybrid’ methodology may be produced that takes relevant aspects from several strategies.

## Traditional Methodologies

Traditional methodologies follow an underlying principle of a rigid development structure in which developer(s) follow a predetermined strategy in separate stages. Most traditional approaches treat each stage’s completion as a prerequisite of moving on to the next.

Heavyweight methodologies are considered to be the traditional way of developing software. These methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing and deployment. Heavyweight methodologies require defining and documenting a stable set of requirements at the beginning of a project.  
- M. A. Awad - A comparison between Agile and Traditional Software Development Methodologies (2005)

### Waterfall

Has an organised progression between each stage of the development process where a set of requirements must be met at each stage in order to progress to the next. The stages are typically ordered Analysis, Design, Implementation, Testing, Deployment and Maintenance in that respective order. A variation of this process includes returning to previous stages if problems occur that were not foreseen and then working back ‘down the waterfall’ stage by stage.

Analysis

Design

Implementation

Testing

Deployment

Maintenance

### Unified Process

There are several methodologies based on a Unified Software Development Process. This Unified Process is an iterative and incremental development technique which involves the four stages:

* Inception – A short period in which preliminary investigation is done into the project’s cause, scope and use-case and project schedules and cost estimates are drawn up.
* Elaboration – Product requirements, risk analysis, concept of design and more finely detailed plans are determined and assessed in this phase. Anything that is a prerequisite of the construction will occur here.
* Construction – The actual implementation of the system. A functional system should be available by the end of this process.
* Transition – The system is released to its users and further refining of the system can be done as a result of the feedback from the users.

### Spiral Model

A spiral model is a risk-based development strategy which identifies these potential risks and attempts to minimise and reduce them. Each cycle of the spiral process involves producing prototypes to ease the identification and analysis of potential risks as well as allowing the product’s end-user to steer the development in the desired direction.

There are four primary stages of a spiral model:

* Target setting – Define goals for the current stage.
* Risk assessment and reduction – Risks are identified, assessed and an attempt at reducing them is made.
* Development – This can be the creation and advancement of a prototype that is built-up over the process of the spiral to finally resemble the end product.
* Planning – Evaluation of the project is made and the next spiral stage is planned out.

## Agile Methodologies

Agile methodologies are commonly known to be ‘evolutionary’ and ‘adaptive’ in that they respond to changes throughout the product development and are flexible with their approaches to particular stages. They often rely upon an early product implementation delivery which is built on continuously.

**Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

* + Individuals and interactions over processes and tools
  + Working software over comprehensive documentation
  + Customer collaboration over contract negotiation
  + Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more  
**<http://agilemanifesto.org/>**

## Extreme Programming

This strategy is adaptive to user requirements in that it employs short development cycles and continuous feedback to incrementally develop the product in an evolutionary manner. This grants the benefit of being able to direct and steer the project to a more accurate degree as opposed to an approach that uses a specific implementation phase which imposes a greater risk upon the product exactly matching user requirements. Extreme Programming follows roughly a dozen practices to ensure its effectiveness which include strict rules such as limiting work hours to 40 per week and always involve a real, live user on the team who is available full-time to answer questions. (Beck, 2000)

## Scrum

Scrum is most often used amongst teams of developers and bases its core principles upon frequent communication and discussion. Another of its fundamental principles is that the user’s requirements can change which could cause unforeseen challenges at any point. It is also known to be an iterative, incremental process and has specific intervals at which meetings must occur and ‘sprints’ (30 days of adapting to environmental variables and adapting the software) end.

Because of the way Scrum is structured it is favourable for development teams and long or continuous projects, thus, making it ill-suited for this project. (Awad, 2005)

## Rolling Wave Planning

Rolling Wave Planning introduces the concept of adaptive planning whereby a ‘wave’ signifies readjusting and refocussing the plan based on project clarity and understanding. I.e. As progress is made, ensure the plan is also adapted to suit the current goals. (Larman, 2003) notes that Rolling Wave Planning is a strong strategy of adapting to variables that can change in the project and integrates neatly into an agile methodology.

## Conclusion

The project’s development will be carried out using an agile approach due to the lack of predictability in the analysis, design and implementation phases. Specifically, obtaining the robot may have significant time fluctuations depending on the results of the analysis and whether one can be bought or if it will need to be built.

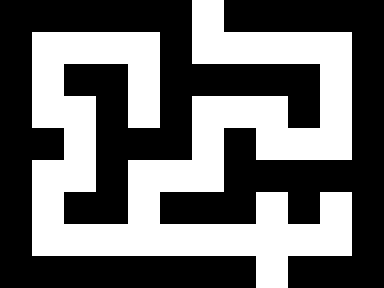
Agile was also chosen to get a semi-functional prototype rolled out as soon as possible to get potential hurdles with the robot’s core functionality cleared up as opposed to them occurring in the later stages of the project when time constraints are limiting. Then the robot can be further developed and stretch goals can be worked towards.

Rolling Wave Planning will be used to reanalyse and reassess the project’s progress, goals and strategy in several waves. These are outlined in the gannt chart.

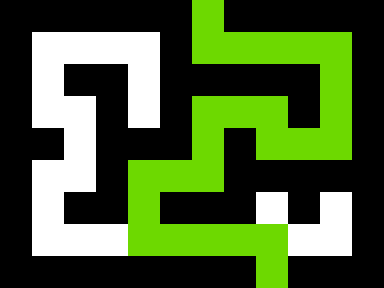
# Pathfinding and Navigation

The difficulty of solving a maze depends upon several factors - the most important is perspective. Solving a maze from a top-down, external perspective where the maze’s entirety can be viewed is considerably less complex than solving from a first-person perspective. For example, having somebody solve a relatively small maze drawn on paper can be completed in a matter of seconds, however, putting somebody in a physical maze of the same design would prove to take a much longer time for them to solve.

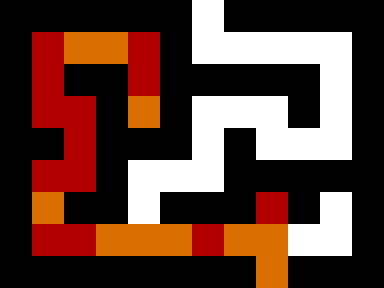
**Experiment**

To understand exactly how much difference perspective can make the following experiment was conducted on a friend, the participant. This experiment is similar to one that (Ferrari, et al., 2001) use to show how external references and note-taking can aid maze-solving from a first-person perspective. A simple maze was drawn on paper, as shown.

The participant was asked to draw the route from the bottom of the maze to the top with a pen as quickly as possible. The route in green is the route he drew and this took approximately 3-5 seconds.

For the second stage of the experiment the participant was asked to solve the maze again without looking at it. He would state which direction he wanted to go in the maze by stating ‘up’, ‘down’, ‘left’ or ‘right’. If the action he stated would be possible he would be told ‘Ok’ and if not (action blocked by a wall) he would be told ‘Wall’. The following route in orange shows the path the participant took with all the times he was blocked marked in red:

This process went on for 2-3 minutes before the participant gave up. Thus, showing that navigating a maze from a first-person perspective is a more difficult task than from a third-person one.

When the participant was asked exactly “how” he solved the maze in the first task, the answer was that he could just see the path to the finish. (Crowe, et al., 2000)’s research using cognitive pattern studies suggest that when humans solve a maze from such a top-down view that they sub-consciously perform several operations based on observations to quickly find a solution. In essence this occurs as a brute-force method where the solver will quickly run through as many routes as possible until they find one that gets them to the end, and in a larger drawn maze may become more evident that this is how a human will generally solve a maze.

**Maze Mapping and Analysis**

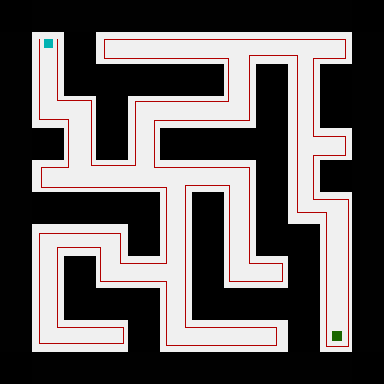
As the task for our robot is to travel from the start to the end point in the maze via the quickest route our robot must first learn all the possible routes there are to achieve this. This cannot be done unless the routes are known which means an initial period will be needed where the robot maps out the maze to get a ‘third-person’ perspective of how it should travel the maze. It must visit all possible paths in the maze to be able to calculate the best route, this means a mapping algorithm will need to be implemented.

**Mapping**

**Left Side or Right Side**

A well-known approach to finding the exit of a maze is to stick to the left or right side and just follow it until the exit is reached. In a maze with no exits to the outside this would be a suitable way to map out the entirety of a maze. However, (Ferrari, et al., 2001) state that we can only do this under the following two conditions:

1. *“When the maze is flat, and has both the entrance and exit placed along its perimeter*
2. *When the maze is flat, and the entrance and exit are points arbitrarily chosen anywhere in the maze, where the latter doesn’t contain any loops. That is, it doesn’t contain multiple paths that connect any two points”*

**The maze specifications state that the start and end points are allowed to be placed anywhere and that loops are also valid. This means that the left/right side rule will not be of use here.

They go on to state that:

“When you cannot apply the rule previously stated, you rely on two strategies:

1. Executing random turns
2. Tracking your route”

As the first approach is random it could potentially take a long time depending on the complexity or nature of the maze. The second approach would yield more fruitful results and the robot will have the capability to do this with its own memory and storage system.

As the maze is split into a grid with each tile being a square of the grid of the same size it makes it possible for the robot to track which tiles it has been to and which ones it has not. The following is an example of how this could work:

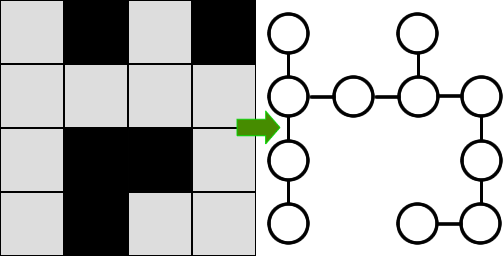
|  |  |
| --- | --- |
|  | This is our maze with our robot (orange square) inserted at a dead-end. During the mapping process it will not matter where the robot begins as its job is to visit every part of the maze so start and finish points are irrelevant here. |
|  | It starts by moving in any direction. This could be prioritised by moving forward, left, right in respective order depending on which is blocked by a wall. In this case, it moves forward, memorising each tile it has visited (green squares) until it reaches a point where it is surrounded by more than one non-wall. As |
|  | It marks this tile in its memory (denoted by the blue square in this example) and then repeats its movement directive by checking if the tile directly in front of it is blocked first. It isn’t, so it continues moving forward until all its sensors report walls. This means our robot has hit a dead end so it must trace its route back to the last blue tile it visited. |
|  | To make the process of journeying back to the last decision tile (blue) easier it can make a note of its directives as it executes them. In this case, it travelled *forward*, *forward* from the decision tile which means to get back to this tile it must travel *back*, *back*. |
|  | The robot now travels in the next direction that it hasn’t visited. Again, in the event that there is a choice between multiple paths it can prioritise directions. Alternatively, it can simply choose randomly between the free paths available as it will make no difference to the efficiency of the mapping process. |
|  | Using this process of marking choice tiles as it traverses the maze allows it to know there are still unexplored tiles meaning that whilst these blue tiles still exist that the mapping process is incomplete. |
|  | This process continues until it has no more choice tiles to return to, meaning the mapping stage is complete and the robot now has a memorised version of the map stored, ready for processing and analysis. |

**Navigation**

Once the mapping process is complete, the Micromouse will have a virtual model of the maze stored in its memory. With this data it can calculate the fastest route from one point to any other and translate the route into a series of actions it must perform in order to traverse the path to the finish.

To apply a path finding algorithm the maze’s tiles must first be stored in a manner which represents each individual tile and its individual properties. Each tile will typically contain information about the following:

* Its position in the maze (X, Y co-ordinates)
* Connections - other tiles it is next to
* Whether it has been visited

Other properties may be required depending on the path finding algorithm used. Graph theory will be used to represent the maze and will allow standard graph theory techniques and algorithms to be applied to the scenario. (Millington, 2009) Explains that this will entail translating the maze's structure and information in real space into a logical, digital form that can be read and stored by the computer. The process for this is defined as follows:

* Each tile in the maze is represented as a graph node
* When the Micromouse travels from one graph node to another, it adds each tile to one another's 'connections'
* When every tile has been visited, the maze is now fully mapped and can be virtually envisioned by the Micromouse in the form of an undirected graph

Then path finding and navigation algorithms can be applied to the graph to find the fastest route through the maze between any given tiles. - (Kühl, 1996)

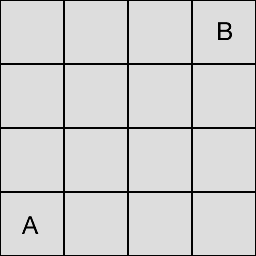
**Algorithms**

Depth-first search (DFS) employs the strategy of traversing as deep as possible into the graph, then recoiling back up and repeating the traversal in a different route. This would be an optimal solution for the grid above as B is as far away from A as it can possibly be and the search will discover the destination in the first traversal. However, this algorithm could be inefficient if B were closer to A and the branches searched first didn’t connect to the destination.

Breadth-first search (BFS) achieves the opposite in that it explores all nodes connected to it first, then all nodes connected to each of its children, in turn. This is efficient if the destination is close to the starting position, but unlike depth-first search, it can be costly if the destination is located far away at the deeper ends of the branches.

Improving from a ‘blind-search’ strategy are cost-based algorithms which take edge cost into account. For example, if we wish to travel from position A to position B via the quickest route, and we have two routes: One which involves travelling up a hill and the other which doesn’t, both of which have the same number of tiles. A cost based algorithm such as Dijkstra will reveal the latter route to be quicker whereas a blind-search one such as DFS or BFS may find the former as the quickest route. However, our maze will not involve direct tile-to-tile costs, therefore Dijkstra would serve no more efficient than a BFS.

**Heuristics**

In algorithms and problem solving in general, a heuristic is a means of guiding decision making in a certain direction by determining which choices are more favourable using predefined information or information collected from previous experience in the current problem - (Weise, 2009).

In path finding, a heuristic can be used based on whether the target destination’s position is known. For instance, if our entity starts at A, and its target destination is B, it cannot decide which direction to start searching first without knowledge of where B is on the grid. Two of the most well-known search algorithms that do not use heuristics are depth-first search and breadth-first search.

There are several variations of heuristics used for path finding, each with their own merit depending on the type of movement and steering behaviour possible. These are:

|  |  |
| --- | --- |
| **Heuristic** | **Description** |
| Manhattan | Distance between two positions using right-angle edges. I.e. staircase style pathing |
| Euclidean | Distance between two positions using a single straight line |
| Octile | Distance between two positions via 45° diagonal edges only |
| Chebyshev | Distance between two positions via any angle which is a multiple of 45°. Essentially combining Manhattan with Octile |

To make use of these heuristics an algorithm that takes heuristic cost into account is required. The most common algorithm for heuristic path-finding is known as A\* (pronounced A-star). A\* is simply Dijkstra’s algorithm using heuristics.

**Movement**

Moving forwards, turning, and diagonal movement?

**Basic requirements of the robot**

Needs to sense if wall in front, left side, right side.  
Needs to be able to check coloured marker beneath it.  
Must be able to move forward and rotate 360deg.

**Detailed requirements**

IR sensors, optical sensor?, battery pack, CPU, two wheels etc

# Bibliography

Awad, M. A., 2005. *A Comparison between Agile and Traditional Software Development Methodologies,* s.l.: s.n.

Beck, K., 2000. *Extreme Programming Explained.* s.l.:Addison-Wesley Professional.

Crowe, D. A., Averbeck, B. B. & Chafee, M. V., 2000. Mental Maze Solving. *Journal of Cognitive Neuroscience,* 12(5), pp. 813-827.

David, W. M., 2001. *Analysis of Micromouse Maze Solving Algorithms,* s.l.: Portland State University.

Ferrari, M., Ferrari, G. & Hempel, R., 2001. *Building Robots with LEGO Mindstorms.* s.l.:Syngress.

Kühl, D., 1996. *Design Patterns for the Implementation of Graph Algorithms,* Berlin: Berlin Technical University.

Larman, C., 2003. *Agile and Iterative Development: A Manager's Guide.* s.l.:Addison Wesley.

Millington, I., 2009. *Artificial Intelligence for Games.* 2nd ed. s.l.:CRC Press.

Mishra, S. & Bande, P., 2008. *Maze Solving Algorithms for Micro Mouse,* s.l.: IEEE.

Sadik, A. M. J. et al., 2010. *A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory,* Dhaka, Bangladesh: Islamic University of Technology.

Sharma, M., 2009. *Algorithms for Micro-mouse,* s.l.: Kaizen Robeonics.

Weise, T., 2009. *Global Optimization Algorithms - Theory and Application,* s.l.: s.n.