|  |  |
| --- | --- |
| **Class** | **Description** |
| Controller | Controls the flow of the program and has the main loop inside. |
| Grid | This class holds information about the cells in the grid. |
| GridCell | This represents a cell on a grid, or a tile in a map. Has an x and y value to represent its position. Also contains information about whether or not it is passable/traversable. |
| GridCellEdge? |  |
| GridDirections? |  |
| Robot | A representation of the LEGO mindstorm robot that will be used to move around the grid and discover the map using the attached scanner. |
| RobotScanner | Represents the physical scanner used on the LEGO mindstorm robot. It is used to “scan” the neighbouring cells to find out information about them. |
| Vector2d | Represents a location in 2D space. Used to track the position of the robot etc. |

# Class Descriptions and Explanation

## Controller

|  |  |
| --- | --- |
| **Attribute** | **Description** |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Method** | **Description** |
|  |  |
|  |  |
|  |  |

## Grid

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type** |
| cells | A list of cells that appear on the grid.  This has to be a list as the grid needs to grow over time to allow the robot to find new cells as it explores. | GridCell list. |
| cellSize |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Method** | **Description** |
|  |  |
|  |  |
|  |  |

## Robot

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type** |
| gridPosition | This attribute represents the robot’s position in relation to its local world. | Vector2d |
| rotation | The robot’s current rotation – this could instead be replaced with a facing direction and the rotation method could use this to work out the best way to rotate. | float |
| sensors | An array of simulated sensors used to get information about the world around the robot. | RobotSensor array |
| speed | The speed in which the robot can move around the world. | Float |
| worldPosition | The position of the robot in pixels rather than grid space. This attribute is used to actually move the robot. | Vector2d |
|  |  |  |

## Model

Includes the virtual data representation of classes such as the grid and the robot.

## View

Will include the graphical representation of the classes and will draw to screen the information provided by the model layer.

## Controller

Will include things such as the pathfinding algorithms that will modify the model layer and move the data around as necessary.

## Simulation walkthrough

At the start of the simulation the robot will make the assumption that the cell it is currently positioned within is traversable and will add it to the local representation of the map.

From the starting position the robot will scan each of the four possible directions and add the neighbouring cells to the local map representation.

Based on the current pathfinding algorithm the robot will select the next cell to move to and will begin moving there.

Once in the new cell the robot will proceed to scan the remaining three cells – only three are needed because the robot already knows about where it comes from.

The robot will continue to move to new cells and adding them to the map representation until no more child cells remain.

Once the whole map has been explored the robot will move back to the original starting position – not sure about this part yet.

## Measurable

Each of the cells in the local representation will keep track of how many times they have been visited.

The total number of moves needed to fully explore the map will also be recorded.

The duration of the entire exploration will be tracked.

## Comparison

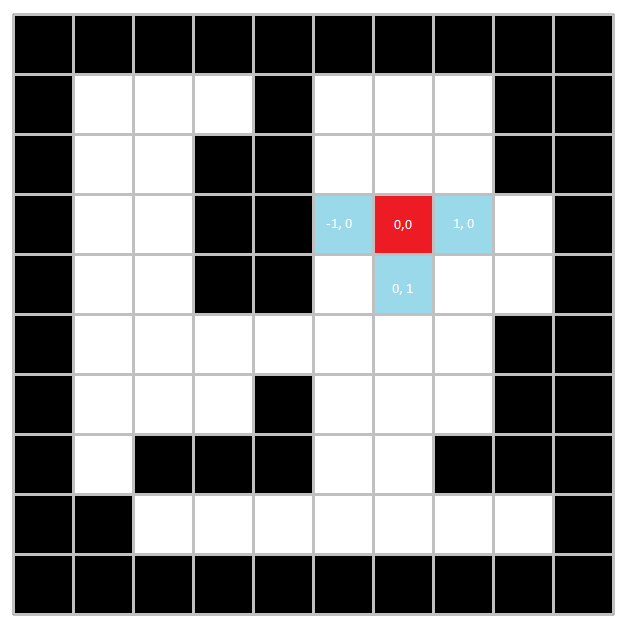
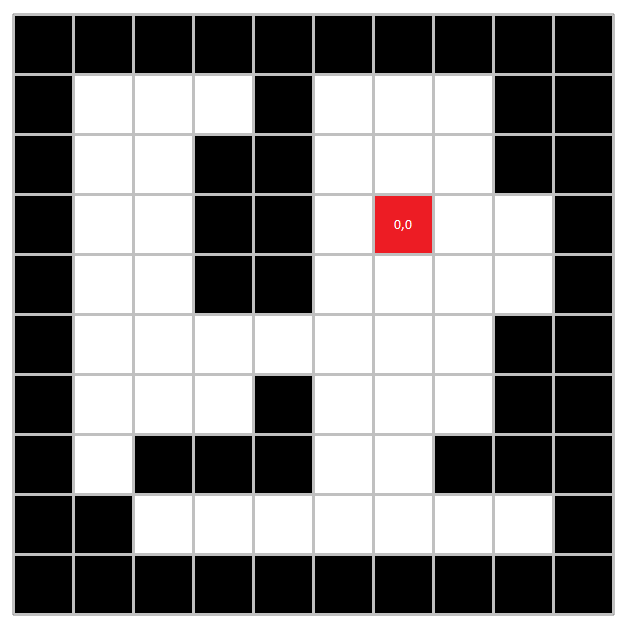
Another simulation, one without the physical limitations of a robot, will also be performed to try and find the “optimal” path through each of the maps.

This will allow a comparison of each of the algorithms to be made against its non-physical counterpart.

## Grid Representation

The grid is perhaps best represented through the use of a list containing GridCell objects. The choice of a list instead of simple a multidimensional array means that the map can grow and shrink as necessary which will be required while the robot is exploring as the total dimensions of the map will be unknown to it.

Each of the grid will be made with coordinates based on the robot’s local position, as this is the only information available to it. Thus the robots starting position will be position (0, 0) and everything else relative to this.



Each of the cells within the grid will contain information about whether or not the robot is able to move that location of not. This information will be original obtain from the robot’s scanners.

### Grid Sizes

A number of different sizes of grid will be created to test the scalability of each of the algorithms. This will help determine which of the algorithms is best suited to which kind of environment. Not only are different sized grids going to be tested by different shapes as well, tall and wide grids will each be tested to determine if the shape makes a difference.

### Obstacles

Each grid size will have a number of variants, both with obstacles and without obstacles to determine how each algorithm deals with non-traversable objects. There is a chance that this will not make much difference in some cases however, the problem of the robot actually having to physically be at a location to scan the neighbours may provide interesting results when faced with obstacles.

## Robot Representation

The robot will be able to move in all directions, however it will only ever need to move forward and to be able to rotate 360 degrees. This will allow it to always be facing forward when it moves.

The robot will also be equipped with 3 scanners on the front, left and right hand sides. These scanners will be able to “scan” the ground and make a decision about whether or not there is first a cell at the scanned location and secondly whether or not the cell is traversable. Once the scanner has provided the robot with this information the robot can then add each of the cells to the local grid representation. The robot will also only add cells that currently do not exist within its representation of the grid.

### Robot Moving

Before moving the pathfinding algorithm will determine which cell the robot should move in and provide the robot with details of this. The robot will then proceed to move to the designated position.

The robot’s world position is updated until it reaches the starting position + the grid cell size in the direction the robot is travelling. Once the robot reaches the correct world position, its grid position can be updated.

Once the robot’s grid position has been updated the robot will then tell the pathfinding component that it has finished movement and is awaiting the details of the next cell it should move to.

## Choice of Programming Language and Environment

The two main options for programming languages is between C# and Java. C# is an option due to the previous experience using it, and the visual studio environment, to develop programs and applications.

However, as this dissertation is about simulating the LEGO mindstorm robots before providing a physical implementation of the tests, the Java language and LeJOS framework have also been considered. This framework provides a replacement firmware for the LEGO robots and also provides a suitable API for communicating with the robot through the Java programming language.

One of the disadvantages of using the Java language and most likely the eclipse IDE is the lack of familiarity with both of these, which would perhaps increase the development time.

It could possibly be said that if the C# language was chosen to create the simulation that the results may vary when it comes to creating a physical simulation in further studies. However, this could be well outside of the concern of this project and might not be a problem at all.

A benefit of using the visual studio environment is all of the built in diagramming tools and also easy to use plugins for features such as version control.

## Choice of Version Control software

The main option for version control is the GitHub solution. It provides free access and has a visual studio plugin to allow control from within the IDE.

## Choice of Algorithms

There are only very few algorithms that are suitable for an explorative search, mainly depth-first search and breadth-first search and derivatives of the two.

As there are no weightings involved in each of the cell edges, the need for an algorithm such as Dijkstra’s would be overkill.

## Program Flow

1. The grid instance is created and setup with obstacles etc.
2. The robot object is created and passed the instance of the grid – for scanner reference only, the Robot is still “unaware” of the layout of the grid.
3. The pathfinding instance is setup and passed a reference to the robot.
4. The pathfinding object then begins looping until all the grid is explored.