The University of York          Department of Computer Science

**Submitted in part fulfilment for the degree of BEng.**

# Assessing the Difficulty of a Generated Path-Planning Problem

Stephen Lewis Webb

April 5th 2016

Supervisor: Dr. Rob Alexander

Number of words = 0, as counted by wc -w.
This includes the body of the report, and Appendices **??**, **??**, but not **??**.

**Abstract**

Here is where the abstract goes.

Dedication.

# Contents

*Contents*

# List of Figures

# List of Tables

# Part I

# Preliminaries

# 1 Introduction

## 1.1 Motivation

Autonomous robotics are increasingly finding application in unpredictable, dynamic environments such as offices, road networks and shopping centres. Human oversight in these situations may be minimal or unlikely. Furthermore, the decisions taken by robots in these applications may well be safety critical - consider, for example, the actions taken by an autonomous car when an unseen pedestrian runs into the road ahead. Such robots must be carefully tested so that we are confident that they may be trusted to operate alone in these environments.

The problem of exploring and navigating the environment has received much work, producing well known methods such as SLAM. Emphasis is now moving towards testing such navigation algorithms so that we may be reasonably confident of the safety of both the robot platform and those in the environment around it. We must be confident that the robot will act safely in the situations it may find itself in during the course of its duties.

There is increasingly argument that traditional testing methods alone are not sufficient to prove the safety of an autonomous robot [ref Rob]. Consider, for example systems coverage - testing each system component at least once. This may not reveal faults in situations which are not addressed by any of the system components. Likewise, while we might test using requirements coverage, we cannot be certain that the stated requirements do not neglect to identify some unforeseen circumstance. Likewise, scenario coverage may have also neglected to include a scenario which such circumstances.

We must, in addition to traditional testing, aim to test the robot in as many diverse situations as possible. However, given any high level situation description - for example, navigating around a person in a room, we can generate a practically infinite set of test cases - every possible position of the person, every possible configuration of the furniture, every possible combination of every such variable. Clearly it is intractable to test every possible situation - instead we must aim to test a diverse

enough range of different situations to ensure we are confident in the algorithms and platform of the robot.

## 1.2 Background

## 1.3 Project Aims

While the generator is capable of evolving diverse maps, no attempt is made to assess the difficulty of navigating such maps. Doing so would allow the system to prioritise more difficult maps, covering more trivial situations implicitly.

This project therefore aims to improve the classification carried out, building upon Wei's generator. To do so the factors which cause a map to be difficult to navigate will be explored and identified. Following this, the most promising will be selected and a method of measuring each defined. This measure will be implemented into the generator. The classification taxonomy will be extended to include these new measures, allowing selection of maps to take them into account. Finally, the extended taxonomy will be evaluated against the original taxonomy.

It is hoped that these extensions will improve the set of maps generated, resulting in both more diverse and more difficult maps. By increasing diversity, we cover more situations and potentially uncover more faults. By increasing difficulty, we avoid testing trivial situations and improve confidence in the results of our testing - if difficult situations are successfully covered, we may be confident that the more trivial maps would also be successfully covered. As a result, the time and cost of testing an autonomous robot may be reduced whilst improving our confidence in it's ability to act safely without human oversight.

## 1.4 Document Structure

This project is divided into the following structure.

**Chapter 2** reviews existing literature on the testing of autonomous robotics and description of Path Planning Problems, before focusing on the factors that make an environment difficult to navigate.

**Chapter 3** describes the problem this project addresses, defining the task to solve and the set of requirements to be delivered.

**Chapter 4** sets out the design and implementation of the simulator and

extensions made to the generator, including rationale for design decisions made during the project.

**Chapter 5** evaluates the extensions made to the generator utilising a set of test agents and the simulator.

**Chapter 6** presents conclusions drawn from the evaluation of the changes made to the generator.

**Chapter 7** suggests future work and improvements to the generator.

## 1.5 Statement of Ethics

# 2 Literature Review

## 2.1 Introduction

This section begins with review existing literature on testing methods for autonomous robots and path planning problems. The difficulties faced by these methods and representations and the potential improvements identified and discussed. Following this, literature on the factors of difficulty used to test current robotics (for example in competition scenarios) is discussed and a likely candidate for the goals of this project identified. Finally, this candidate is investigated in greater detail.

## 2.2 Testing Autonomous Robots

Testing autonomous robotics is a very difficult task. The environment in which they are to operate is often complex and dynamic; in addition many of the domains to which autonomous robots are being applied are safety critical. An example of such an application is autonomous road vehicles, which face the challenges posed by the roads and fellow road users and must behave in a safe manner. To consider such a vehicle safe, we must have confidence that it is capable of acting in a manner which does not endanger itself or fellow road users in any situation it may find itself in. In [1], Umerson et al. tested each algorithm driving their autonomous vehicle offline in simulation or via data replay, before moving onto on-vehicle testing. Similarly, Google's autonomous vehicles have been tested for thousands of hours[2]. On-vehicle testing, however, presents the possibility that something could go wrong, potentially endangering the vehicle, its occupants, and other road users. We might test for confidence offline, like Umerson - but hand crafting each simulation map is time consuming and restricts the total range of testing which may be carried out. Furthermore, faults may not become apparent until the system as a whole is tested on-vehicle outside of simulation.

Indeed, there is increasingly argument that new approaches to testing are required to sufficiently cover the range of situations a robot may

encounter while working autonomously. In [3], Alexander contrasts his work focusing on coverage of a range of situations with other methods such as requirements and systems based testing. Consider requirements testing - while we may successfully pass tests against stated requirements, this potentially misses failure cases which the requirements missed. Likewise, systems testing cannot identify faults that arise as a result of an external factor. Alexander argues that we must, in addition to other forms of testing, also test over a wide range of situations (for example, navigation of a robot in a room), the factors of such a situation (furniture, people, etc. in the room) and possible combinations and configurations of such factors (layout of furniture, position of people..). The aim of such testing is not to test every possible situation arising from these factors but to cover a wide and diverse range - and in doing so convince ourselves that the robot under test is safe. Hence, a large number of diverse testing scenarios are desired.

## 2.3 Path Planning Problems

A Path Planning Problem (PPP) is, in its simplest form, a specification of an environment to navigate and a description of its features, such as the size of the environment, location of obstacles, and so on. Richer aspects such as 3D terrain and surface materials might also be included. It is upon this environment that the robot or algorithm may be tested. Such specifications may range in complexity and features. In addition, the PPP may also specify the initial configuration of the robot under test. The problem is then to successfully find the (optimal or near optimal) range of motions from the initial position to the goal position, avoiding obstacles. PPPs can be used to test autonomous vehicles, algorithms, industrial robotic arms and so on. Indeed, they are not limited to simulation - one could use the specification to set up an arena with physical robots and obstacles in which to test the robot.

PPPs may be used for traditional testing for a robot or algorithm. As an example, consider requirements-based testing of an industrial robot arm. If a requirement states that the arm must be able to pick up an object and move it to another location, one could produce PPPs describing a simple environment with the object and target location and the configuration of the robot arm (before picking up the item, after picking up the item, for example) and then simulate the robot to ensure it moves from each position to the goal.

As discussed above, a chief difficulty in producing PPPs to test robots is the cost and time required to develop a large number of diverse problems, particularly if we must hand-craft them. In [4], Ashlock puts forward a PPP generator for simple 2D grid problems, based upon a genetic algorithm. He evolves the produced PPPs to maximise three measures - minimum number of turns and advances moves required to solve the PPP, and the sum of the minimum turns and advances, as calculated by a dynamic programming algorithm (DPA). This DPA also implicitly discards uncompletable PPPs. This approach allows for a large number of PPPs to be evolved in short order.

However, as noted above, it is not enough to test just a large number of situations - if they are all too similar (consider situations in which the only difference is a slight rotation in a piece of furniture) then very little practical benefit is gained from the testing, even over a large number of such tests. Thus, Ashlock also seeks to provide diversity in the final collection of PPPs. He makes use of a neighbour joining taxonomy inspired by the taxonomic procedures seen in biology. This requires taxonomic characters - measurable, computable qualities describing a PPP - to be extracted. Ashlock makes use of the three fitness function characteristics, alongside the total number of obstacles in the PPP grid as his four taxonomic characters. Finally, using the UPGMA clustering method, these taxonomic characters produce a tree of PPPs where like problems cluster in branches. Thus, by selecting PPPs from many branches, ignoring closely related nodes, a diverse collection of PPPs may be produced. As noted by Wei, under Ashlock's simple taxonomy it is still necessary to observe the diversity of a set of PPPs manually as the PPPs are only separated into groups but the diversity in each group is unknown[5, chapter 8, p. 66].

In [5], Wei reimplements Ashlock's work and attempts to evaluate and improve upon it. Through his testing, Wei shoes that Ashlock's generator produces superior PPPs compared to those produced by a random PPP generator by simulating robots with varying levels of completeness in their path planning algorithms. In all cases, the test agents had a lower success rate on Wei's PPPs, showing that his PPPs are better suited to exposing their flaws. In addition, Wei provides optimisation to the generator by pruning PPPs in which the goal is unreachable during a mating event, ensuring only completable PPPs are generated. This prevents the genes of the uncompletable PPP entering the pool and improves the time taken to generate a large number of PPPs. Wei goes on to show that this change also improves the PPPs themselves, again through simulating

robots and comparing the success rates against Ashlock's PPPs.

Ashlock and Wei's work provide a way to select diverse PPPs via clustering and a method to quickly produce a large number of such PPPs. However, neither attempt is made to measure how difficult these PPPs are to traverse - while a robot with perfect information of the map will clearly complete and PPP with a reachable exit via the optimal route, Wei's work shows that less capable robots often fail and become stuck. Indeed, Ashlock's fitness function would maximally reward a completely canalised PPP where the obstacles force the robot along one path in a manner that maximises the number of turns and advances the robot must make. Clearly, such a map would be time consuming but not actually be difficult to traverse, as there is no opportunity to become lost - the only choices are to proceed or reverse. Thus, the faults of a less capable robot are not likely to be discovered.

By assessing the difficulty of a PPP and applying this measure both to the fitness function and as a taxonomic character, the PPP generator put forth by Ashlock and Wei could be improved. We could evolve for more difficult PPPs as well as clustering similar difficulty PPPs together, allowing the user to select both diverse and difficult PPPs to solve during testing. As a result, we could improve our confidence in a robot under test by testing increasingly difficult situations. In addition, by testing difficult situations we can be confident that the robot can behave appropriately in more trivial situations.

## 2.4  Factors of Difficulty

In order to begin assessing the difficulty of a PPP, we must first understand the factors that make an environment difficult to navigate. As autonomous robotics become more prevalent, there are an increasing number of attempts to provide standardised tests for both the algorithms and the platforms, as well as competitions environments from which we might draw some of these difficulty factors.

As an example, consider RoboCup Rescue [6], a competition in which teams of robots attempt to carry out various tasks in a disaster environment, ranging from location of survivors to fighting fires and others besides. The characteristics of the Rescue competition domain seek to simulate a disaster and challenge the teams; robotics must be able to work in a large heterogenous team, planning in the long term and collaborating emergently in a situation that demands sharp real-time response.

In addition, the environment is potentially hostile due to debris and fire and information is missing, partial or incorrect. As such, completion of goals often requires exploration and robust planning to react to the unknown and dynamic environment.

In [7] Jacoff et. al. approach their evaluation of difficulty from a different angle to RoboCup; rather than a simulated situation they identify "fundamental elements of mobile robot autonomy" and seek to challenge these elements. These elements are locomotion, sensory perception, knowledge representation, autonomy from the operator, and collaboration. As such, they designed three arenas of increasing difficulty. Challenges include mazes, locomotion hazards, debris, sensory confusion from the environment and dynamic environments which alter explored areas of the map.

Ashlock has also worked on procedural generation of maze-like maps, taking a similar genetic approach to that used by his PPP generator [8]. Here, his difficulty factors and fitness functions are the shortest path from entrance to exit, accessibility of 'checkpoints' (simply points in the maze which are used to characterise connectivity and path length) in the maze, number of distinct branching paths, and number and length of dead ends.

We must take into consideration the range of robots and applications. It is not necessary to test all autonomous robots with the same rigour. While an autonomous road vehicle would need in-depth safety testing, simpler applications such as the Roomba household cleaning robot have more relaxed safety expectations upon them. In [9], Jacoff notes that a good testing method must ensure that the "figurative measuring stick is long enough to capture performance at both ends of the available robotic capabilities spectrum, and that it separates performance results in between." Therefore, the chosen measure of difficulty must be robust and adjustable so that it might be applied across a wide range of robots.

## 2.5 Exploration

A recurring theme in the literature discussed in the above section is that of exploration; particularly exploration in situations where information is unavailable or incorrect.Indeed, Wei's paper showed that robots which did not have prior knowledge of the PPP often struggled to reach the goal. This section will therefore focus on the problem of exploring an unknown map in order to develop a measurement of exploration which

might be applied to improve the generation of PPPs by allowing for evolution of 'hard to explore' PPPs.

In [10], Meyer and Filiat argue that path planning calls upon three processes - map learning, localization, and finally planning. Learning requires acquisition and memorization of data via exploration. Localization requires the derivation of the robot's current position, interdependent on learning - one must explore to know the current position, and must know the current position to know where to explore. Finally, path planning (as previously discussed) requires the choice of the course of actions to reach a goal from the current position. This requires the map to be represented in some way that is plannable.

The approach to representation of the robot's environment can be broadly decomposed into two categories - Topographical maps use the characterization of various places in the map by external sensors, storing their relative positions. This approach makes updating the map easier than other approaches as new information arrives and the map is reconsidered. However, localization is often much more difficult and highly sensitive - different locations, viewed from different angles or sensor configurations, may seem very similar and thus difficult to distinguish [10].

The second category is metric maps in which the position of obstacles is stored in a common reference frame (for example by decomposing the world into a grid and marking obstacle grid squares as impassable). A popular approach to such maps is the Occupancy Grid [11]. In this approach, the world is decomposed into a grid matrix and the probability that a given cell is occupied is calculated via sensor readings. A background uncertainty initially fills the map; cells with probabilities near this uncertainty therefore have little or no information gathered upon them. This naturally guides exploration; to explore one must plan to move into an area where the probabilities of the surrounding cells are close to the background uncertainty of the environment.

There are two approaches to updating a metric map; a backward model wherein the probability of a cell being occupied is calculated via sensor readings, and a forward model where the calculated probability of a sensor reading informs the map that would produce such a reading. There is some evidence to suggest the forward model produces a better map, though it is more sensitive to change in the environment and is prohibitively expensive for real time application [12].

Given a representation of the environment, there are many approaches to deciding where to move and explore. In [13], Lee describes several

such approaches. A reactive approach holds little internal state, reacting quickly to incoming sensor information, and is suitable for simple exploration methods such as wall following. Another approach is to simply go to the least explored region. A further approach is to "go where it's interesting", basing decisions on the latest map. This attempts to systematically reduce uncertainty in the map, often using the occupancy grid as a method of describing uncertainty. Under this representation, an 'interesting' area is an area with occupation uncertainty close to the background uncertainty - these regions are unexplored, or have very little data gathered upon them. The robot would therefore path to these areas as the pay-off in information to be gathered is higher.

A further, well investigated approach is SLAM - Simultaneous Localization and Mapping - the problem of placing a robot in an unknown environment and having it produce a consistent map whilst simultaneously locating itself within said map[14, p. 99]. There are a large number of approaches to SLAM, a few of which are open source [14, p. 107]. SLAM is, however, computationally expensive and complicated and therefore likely to be too heavyweight for the purpose of this project.

No approach is without its difficulties, however: one of the most common and most important difficulties faced by exploration and in mobile robotics in general is that of sensor fusion. It is rarely sufficient to explore with only a single sensor; this restricts the 'field of view' in which knowledge on the environment may be gathered and is not robust to difficulties to the sensor in the environment (for example, a laser sensor may struggle with a plate glass door). As such, multiple, varied sensors are used. Sensor fusion is the problem of fusing sensor readings from multiple independent sensors in order to produce coherent knowledge about the external environment. There are many approaches to tackling this; the best known are Dempster-Shafer Theory (DST) and Dezert-Smarandache Theory (DSmT). There is a range of literature dealing with the problem of sensor fusion and applying fused sensor readings to map representations. Another difficulty is communication of the explored map to human operators and fellow robots - In [7], Jacoff et. al. identified such communication as a fundamental element of robot autonomy, requiring that the robot is capable of expressing its maps understandably to the humans.

An additional difficulty is a direct algorithmic result of an uncertain environment - path planning approaches need to be dynamic and capable of adjusting to new information. For example, a path generated from what a priori knowledge is available may, during traversal and explora-

tion, be discovered to be blocked. The system must then adjust to this new knowledge and plan a new path. This difficulty requires the path planning algorithm to be coupled with robot movement such that the search space may be updated and costs reconsidered if it is necessary to take a new obstacle into account.

Some approaches to this problem plan a global path based on a priori knowledge and then attempt to path around discovered obstacles, recomputing the global path if necessary. Other approaches generate a route which seems optimal based on current knowledge and proceeds upon it until the sensors detect discrepancies; a new route is then generated and taken [15]. If a priori knowledge is not available, it may be the case that the optimal route is not obvious until the map is first suitably explored. Finding an optimal solution to PPPs without perfect a priori knowledge is therefore a harder problem, though the robot may benefit in the future from its exploration if the environment is unchanging.

As discussed above, this chosen difficulty factor must be both robust and adjustable so that it might be applied across the range of autonomous robots. By evolving PPPs which favour high exploration difficulty - causing, for example, more obstacles to be placed and in a manner which challenges the exploring agent with dead-ends and multiple choices of routes, we can test capabilities at the far end of the spectrum. Similarly, the fitness function might be adjusted to encourage less difficult PPPs to test the lowest end of the spectrum and the range in between.

# Part II

# Research

# 3 Problem Analysis

## 3.1 Introduction

Testing of autonomous robots is a difficult task and, as they become more commonly applied to real world problems, it is increasingly important that we are confident in their safe behaviour. As demonstrated by the literature review, this requires that we cover as many situations that may be encountered in the robot's work as possible - both those we can predict, and those that may not be obvious beforehand. While hand-crafted test situations provide insight into such behaviour, and can be carefully tailored to challenge weak areas of the algorithms, robot platform, and so on, they are time consuming to create and have no hope of catching failures that we can't predict (and therefore can't provide a scenario for) at a testing stage.

Previous work by Ashlock and Wei provides us with the means to algorithmically generate large numbers of diverse path planning problems as test cases, taking us some way in addressing the issue of testing a wide range of diverse and potentially unexpected situations. However, the simple fitness function employed in generating these PPPs would maximally reward an environment where the test agent is forced along a canalised, winding path which maximises distance travelled and turns made, with no opportunity for deviation. Clearly, such an environment is not a challenging test case - the only choice is between proceeding or reversing.

Further work is required to improve the generated problems such that the difficulty of a generated PPP can be assessed more accurately and evolved for as part of the fitness function. This would allow for selection of diverse test cases of a difficulty suitable to the application of the robot under test. The literature review identifies a range of hazards and difficulties employed by current testing standards and robot competitions; the chief amongst these is exploring an unknown map and successfully identifying the optimal route.

This section begins with a high level definition of the goal and task of this project, defining the terms and measurements discussed above in

more detail. Subsequently, the high level requirements will be laid out and rationalised.

## 3.2 Task Definition

The high level goal of this project is to improve Wei's PPP generator to assess and evolve for difficulty in the output path planning problems. These path planning problems are then to be used to find faults in test robots; hence the improved generator should be more capable at identifying faults by causing more test robots to fail to reach the goal location.

## 3.3 Difficulty

As demonstrated in the literature review, there are numerous ways in which a robot test environment may be said to be difficult. These include but are not limited to locomotion hazards such as rubble, complex collaboration tasks and knowledge representation issues. Currently the PPPs produced by Wei's generator are simple 2D grid environments and do not include such complex tasks or rough but passable terrain. However, a common theme in all of the competitions and standard tests discussed is that of knowledge representation and exploration. Clearly, no matter how full the environment is with debris or sensory hazards, any robot with perfect prior knowledge of the PPP will be able to take the optimal route to the goal. The difficulty actually lies in exploration of the environment, identifying these hazards and planning how best to avoid or handle them such that the best route can be found. Therefore, the measure of difficulty will seek to assess how difficult it is to explore a given PPP.

### 3.3.1 Occupancy Grid

Wei's generated PPPs can be considered as a 2D grid with a defined start and goal position and certain cells occupied by obstacles as described by the PPP itself. A test agent with some sensory capability - for example, perfect 'sight' into immediately surrounding cells - must then explore and path to the goal. A dynamic programming algorithm (DPA) is used to assess the optimal path across this grid.

As discussed in the literature review, the occupancy grid map representation closely mirrors this representation. Exploration progress of a bounded map may be understood by initially setting the entire grid to a background 'noise' occupation certainty. Then, during exploration, visited areas of the map become are updated and their occupation becomes more certain. Thus unexplored areas are close to the background uncertainty.

### 3.3.2 Difficult PPPs

A 'difficult' PPP is one where the goal position and optimal route from the initial position is difficult to identify in full without exploration. As such, the actual path taken may be far from optimal and the optimal route may only be apparent once the goal position has already been found and reached. The test agent, as a result of this deviation from the optimal path, incurs a higher than optimal movement cost to reach the goal position.

### 3.3.3 Measurement of Difficulty

By applying the occupancy grid map representation to the current PPP representation, uncertainty may be included in the DPA assessment of the optimal path from the initial position to the goal. Adjustments to the DPA evaluation function can be made to take into account the reward of exploration - finding parts of the optimal route, or the goal position - alongside the cost of movement. Hence, the DPA may model exploration of the PPP from very little a priori knowledge by a simple agent. The resulting optimal route produced by the DPA may be different from the optimal route as assessed by movement cost with perfect a priori knowledge - the deviation may hence be computed and the cost incurred by the requirement to explore derived. A more difficult PPP will therefore require more exploration and make the route difficult to find, increasing this cost. As a result, larger PPPs and those with a greater number of obstacles will clearly be measured as more difficult than smaller, simpler PPPs.

### 3.3.4 Application and Diversity

The difficulty measurement can be included in the fitness function and as a taxonomic character. Inclusion in the fitness function allows the

| Requirement | Description |
|---|---|
| R01 | The generator must assess a notion of difficulty (as defined above) of a PPP. |
| R02 | The difficulty of produced PPPs must be adjustable. |
| R03 | The generator must be capable of producing diverse PPPs. |
| R04 | The measurement of difficulty must be included in the clustering taxonomy. |
| R04a | No single taxonomic character may dominate the taxonomy. |

Table 3.1: Generator Requirements

difficulty of generated PPPs to be adjusted and tailored to a level suitable for the robot under test. In addition, by using this measure as a taxonomic character, the produced PPPs will be clustered into groups of similar difficulty. Hence the final selection of test problems should be improved in difficulty while still allowing for diversity. This should improve the capability of generated PPPs to identify faults in test robots.

## 3.4 Generator Requirements

### 3.4.1 Rationale

The overall aim of this project is to assess a measurement of the difficulty of a PPP. R01 guarantees this. As discussed previously, a good testing standard should be able to challenge the whole spectrum of capabilities. This is covered by R02. We must also test for diverse PPPs, hence R03. The generator currently assesses diversity via a clustering taxonomy. The new measure must be included in this taxonomy so that it might be considered during PPP selection, hence R04.

## 3.5 Simulator Requirements

The code for Wei's evaluation simulator is not available, hence a new simulator must be developed to evaluate the changes made to the generator.

| Requirement | Description |
|---|---|
| S01 | The simulator must accept and simulate the generated PPP format. |
| S02 | Agents with a variety of capabilities must available for testing. |
| S03 | Simulation results must be recorded to disk. |
| S04 | Simulation must terminate in a finite time. |

Table 3.2: Simulator Requirements

### 3.5.1 Rationale

Wei's work laid out a .ppp file format for generated PPPs. S01 ensures the simulator tool can read these independent of changes to the generator. To gather meaningful results, tests must be carried out over a range of agent capabilities, as set out by S02. In particular, buggy agents with known faults must be available so that the ability of the improved PPPs to detect such faults may be understood in comparison to Wei's PPPs. To allow for further analysis, these results must be stored as set out in S03. As we are dealing with agents with faults, it may be the case that an agent may path in a loop and never reach the exit. In this case the simulator should record the test run as a failure for that agent, as never reaching the goal is a clear fault. This is set down by S04.

# 4 Design and Implementation

This chapter describes the methodology, architecture, design decisions and implementation of both the extensions to the PPP generator and the simulator. First, high level decisions are detailed, such as development methodology and the programming language used. Following this, the architecture of the the project will be discussed. Finally, I will focus on design decisions encountered during the development of the deliverables, justifying choices made.

## 4.1 Methodology

The goal of this project is to extend an existing system with a new set of features, making use of a simulator environment to evaluate these features. The requirements as stated in chapter 3 serve to guide development of these features and the simulator.

Given that the features to be introduced to the system will require a cycle of adjustment, evaluation, readjustment, due to the moving-goal nature of fitness functions, development methods such as Waterfall are not appropriate. Instead, a methodology which suits the incremental nature of these features would be far more suitable.

I identified two such methods; Spiral and Agile. Agile methodology [19] seeks to develop and implement milestones in sprints - for example, a facet of the difficulty evaluation. Each sprint then builds upon the previous one, producing the final product incrementally. Each sprint should leave the generator in a stable state, such that changes may be assessed and the goals of the next sprint identified. In particular, a central tenet of Agile is responding to change; this aligns closely with the way the fitness function would need to be implemented and honed over the course of the project.

Spiral methodology [20] functions extremely similarly; goals are identified and implemented iteratively. However, Spiral focuses heavily on risk analysis to guide development. This can be difficult to assess and would not work well for a project seeking to implement many small

components, as opposed to large scale engineering projects. For this reason, I chose to use Agile.

## 4.2 Programming Language

This project builds upon Wei's generator, which has been implemented in Java. This project therefore uses Java, rather than reimplementing or interfacing with existing code via another language. This allowed me to quickly move ahead with the project. In addition, some Javascript and HTML is used to visualise the output trees of the UPGMA functionality.

## 4.3 Architecture

As discussed in the chapter 3, this project is divided into three components: a simulator for the use of evaluating PPP files, a representative set of test agents, and extensions to the generator to measure and evolve for difficulty in the PPPs. The architecture of each will be detailed below.

### 4.3.1 Component 1: Simulator

I initially chose to focus on the simulator, such that the framework and methodology to evaluate the changes made to the generator was available before I began this work. This would allow me to rapidly evaluate the changes and iteratively improve upon them.

I chose not to make use of a third party simulator framework such as NetLogo and MASON. This is because the PPPs and therefore the environment to be simulated are currently extremely simple, requiring only basic sensor functionality and an extremely simple two dimensional environment. The algorithms required to sense in this grid environment are also simple, requiring basic linear interpolation and line drawing. In addition, much of the environment representation could be produced by reusing Wei's code with minor additions such as the ability to load a PPP from a saved textual representation. Finally, regardless of the simulator, I had to implement the agents and their path finding algorithms.

I therefore chose to implement a simple simulator myself, which assisted me in learning Wei's code ahead of making my own additions and changes to it. I felt this was more valuable than spending time and effort to bring in a more advanced and feature-heavy simulator package as it

would take me far longer to learn how to use it and interface with Wei's code and the PPP representations.

### 4.3.2 Component 2: Test Agents

The simulator required a set of test agents with representative faults such that the PPPs could be evaluated for their ability to find these faults. Seven such agents were produced, detailed later in this chapter.

### 4.3.3 Component 3: Extension of the Generator

As this project is built upon Wei's existing code, the majority of the architecture for this component of the project is already in place. As such, this component will consist of extensions and additions to the current architecture and will seek to avoid structural changes.

Currently, the generator is capable of evolving for PPPs favouring an optimal route with many turns or many advance moves, as measured by the reimplementation of Ashlock's DPA algorithm. PPPs are evaluated at creation, or when they are altered by a mating event or mutation.

As such, I will implement new functions which will be used as part of these evaluations, measuring the difficulty of the PPP. Following this, I will implement a fitness function in the same manner as Wei, which will select high fitness PPPs for mating events and low fitness PPPs for mutation. As such, the generator function as before, with additional methods of evaluation and selection.

The result of these extensions should be that the tournament generator is capable of producing a range of PPPs with diverse 'difficulties' as measured by the difficulty evaluation functions, such that these PPPs improve the fault finding capabilities of the simulator over Wei's original PPPs.

As changes were made to my new fitness function, it may be that the goalposts move and tweaks and additions will be required. As such, I began with a simple measure of difficulty. I then used the altered generator to produce a set of 60 PPPs with several hundred mating events. This number was chosen to ensure generation time was quick; though the number of mating events is small, it could be reasonably expected that results would improve with further events, and thus this allowed for rapid 'prototyping' of the new PPPs. I could then evaluate these PPPs using the simulator and by observation to identify good and
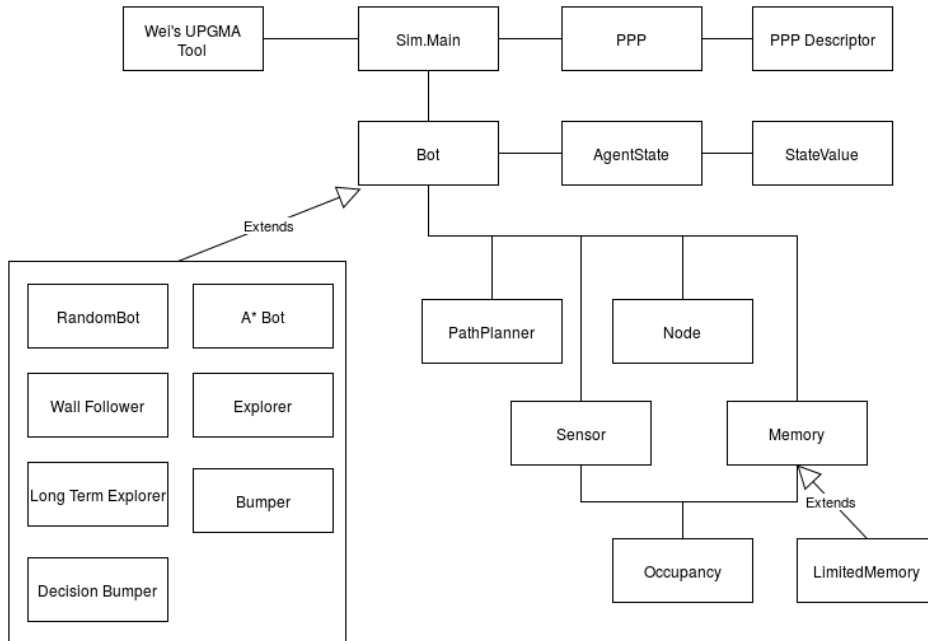
Figure 4.1: Simulator Architecture Overview

bad features. This knowledge could then be used to further improve the fitness function.

## 4.4 Simulator

A high level overview of the simulator architecture is provided in figure 4.1.

The simulator makes use of Wei's existing PPP classes and UPGMA tools to load a textual description of a PPP from disk, representing it as it was when it was first generated. In addition, the simulator makes use of Wei's UPGMA tool and its relevant classes to run the UPGMA algorithm, producing a JSON file describing the resulting tree structure. This is used for visualisation alongside D3.js. In addition, a CSV of the agentâĂŹs results is produced.

Test agents are represented by the Bot class, which defines an interface the simulator uses to have the agent sense, plan and move at each simulation step. The seven subclasses agents each use different planning methods, and have different faults as discussed in section 4.6.

The bot class makes use of several component classes to provide state and functionality to the agents. Wei's AgentState and StateValue classes, used in his DPA implementation, are reused to store the bot's current location, moves made, and so on. In addition, a Node class was produced to describe positions into which an agent might plan to move, whilst a Sensor class provides functionality to describe lines of sight and thus what may be sensed from a location. The PathPlanner class is used to allow generic access to path planning methods such as A*, though not all of the agents make use of these methods. Finally, the Memory class acts as an occupancy grid that is updated over time as the agents move and sense. Plans are made based on this memory, though the simulator tracks movement against the PPP to ensure erroneous plans do not allow the agent to move through walls or out of bounds. This makes use of an occupancy class to describe what a grid cell represents - for example, an empty space or boundary wall.

The LimitedMemory subclass is used to handicap an agent, allowing them a restricted memory space such that visited locations that leave the memory space are forgotten. In addition, the sensor class allows for a noise handicap to be set, which results in erroneous data to be randomly returned when sensing. This chance scales as the noise is set higher.

## 4.5  Test Agents

This section details the test agents produced and their faults. The extensions made to the generator will be evaluated on their ability to discover these faults in the agents in comparison to Wei's existing generator. As described above, each agent may be handicapped with sensor noise and by limiting their memory space. As I intended to incorporate elements of exploration into the measurement of difficulty, I drew several of these agents from exploration tactics as described in the literature review. In addition, I reimplemented agents described in Wei's paper to provide more direct comparisons to his results.

### 4.5.1  A* Bot

This agent uses A* with full prior knowledge of the PPP to find the optimal route. As such, if the goal is reachable, it will always complete the PPP in the optimal number of advances and turns. This agent has no intended fault, and is present as a comparison point to the routes of

other agents.

### 4.5.2 Random Bot

The random bot moves randomly, including potentially staying still. As such, it is easily defeated by most PPPs, with a 4.4% pass rate on empty 20x20 PPPs.

### 4.5.3 Wallfollower

This agent implements a very simple exploration tactic, following a wall and attempting to ensure its next move keeps the wall on the chosen side of the agent. This produces a simple fault - when the walls lead the agent in a cycle, it is not capable of taking alternative routes.

### 4.5.4 Explorer and Long Term Explorer

These agents seek to maximise the number of cells explored, and therefore move to positions that maximise the number of new cells sensed. This leads, for example, to the agent moving away from walls to make use of its full sensor range. The basic explorer does not plan far ahead, however, and may wander aimlessly in fully explored areas. The long term explorer may recognise when it cannot explore further, and will seek to path via A* to a new unknown area. However, this is not very guided and may result in the agent moving far away to explore single cells though there may be areas with a large number of unexplored cells.

### 4.5.5 Bumper and Decision Bumper

These agents are reimplementation of two of Wei's agents. Both move in straight lines until they bump into a wall. At this point, a new direction is chosen at random. This may lead to both being trapped in areas where the only exit is not in a position such that they will path into it. The decision bumper records its directions at each 'bump', and will exhaust unused choices before choosing randomly.

## 4.6 Test Agent Faults

The set of faults in the test agents, while simple, is a worthwhile set as they represent faults which would be faced by real world agents

implementing these simple exploration and movement tactics. If a fault may be uncovered due to an agent only planning one step ahead, then it can be reasonably expected that similar faults might be uncovered if the agent only planned two or three steps ahead (and so on) though they might be rarer. As such, I argue that they represent the 'worst case' of a set of related faults - planning further ahead should result in a better pass rate for a given agent. The faults in the agents described are therefore fairly representative of real world faults.

In addition, as noted in the chapter 2, a common issue facing autonomous robotics is that of accurately sensing the world through multiple sensors, each with different capabilities and viewing positions. This is usually tackled via one of the many methods of sensor fusion. While the agents and simulator implemented are very simple, each having a single sensor which operates by line of sight rules from the agent out to its range, the sensor noise handicap goes some way to better representing real world agents by modelling the imperfections in a real sensor suite. As the PPPs themselves are very simple, I did not consider it necessary to implement a more complex sensor model in the simulator; in particular it should be noticed that sensor noise obscures the measure of difficulty as I could not be sure if the PPP was difficult due to its layout or the effect of sensor noise. However, it should be clear that a noisy sensor should reduce the pass rate of an agent.

## 4.7 Extension of the Generator

As discussed above, extensions to the generator would take place through the addition of new evaluation functions to the PPP class and the addition of new fitness functions to the PPP Generator class which would draw upon these evaluations.

The result of these extensions should be that the tournament generator is capable of producing a range of PPP with diverse 'difficulties' as measured by the difficulty evaluation functions, such that these PPPs improve the fault finding capabilities of the simulator over Wei's original PPPs.

### 4.7.1 Goal Visibility

I began with a simple measure of difficulty - a measurement of the number of cells in the PPP from which the goal position was visible, taking

into account lines of sight due to obstacles. This would be measured with a perfect sensor - that is, one with zero noise and a range capable of sensing the entire environment. Clearly it would be the case that inferior sensors would further increase the difficulty - sensor noise would cause incorrect measurements, whilst reduced range would reduce the number of cells from which the goal was visible. Hence, a perfect sensor provides a base case.

The premise behind this measurement is that PPPs that are difficult to navigate must necessarily place obstacles such that the optimal route is obscured. If this was not the case, the optimal route would be a direct path to the goal position. As there would be obstacles between the start and goal positions, the optimal route must necessarily circumvent these obstructions. If this is the case, then even in cases where the PPP is known beforehand, the route ought to be longer and harder to navigate than a simple, unobstructed route.

The sensor and memory classes from the Simulator were reused to produce these measurements, as the line of sight functionality was already present. Lines of sight are generated from the goal position across the PPP. Each line is then followed, counting the number of new cells explored. If an obstacle is met, the remainder of the line is discarded. This measurement can therefore be applied regardless of goal position. The fitness function then selects for PPPs with lower goal visibility measurements.

An example of a PPP generated under this fitness function is provided in figure 4.2.

After prototyping a set of PPPs with this fitness function, I noted that the generator was encouraged to place obstacles in clusters around the goal position such that the greatest number of cells were obscured. Placing obstacles further away is less attractive, as it leaves many cells on the side of the goal which can still 'see' into the goal. As a result, chromosomes which produce obstacles further from the goal tend to be bred out over the course of the tournament. At the extreme, this resulted in PPPs which were mostly empty with the exception of these blocking obstacles, and thus were easy to traverse.

### 4.7.2 Visibility Magnitude

To attempt to encourage the generator to spread obstacles across the PPP, I added additional measurements of visibility to the function. These measurements would take into account the visibility from each of the four corners and the centre of the PPP, in hope that the generator would
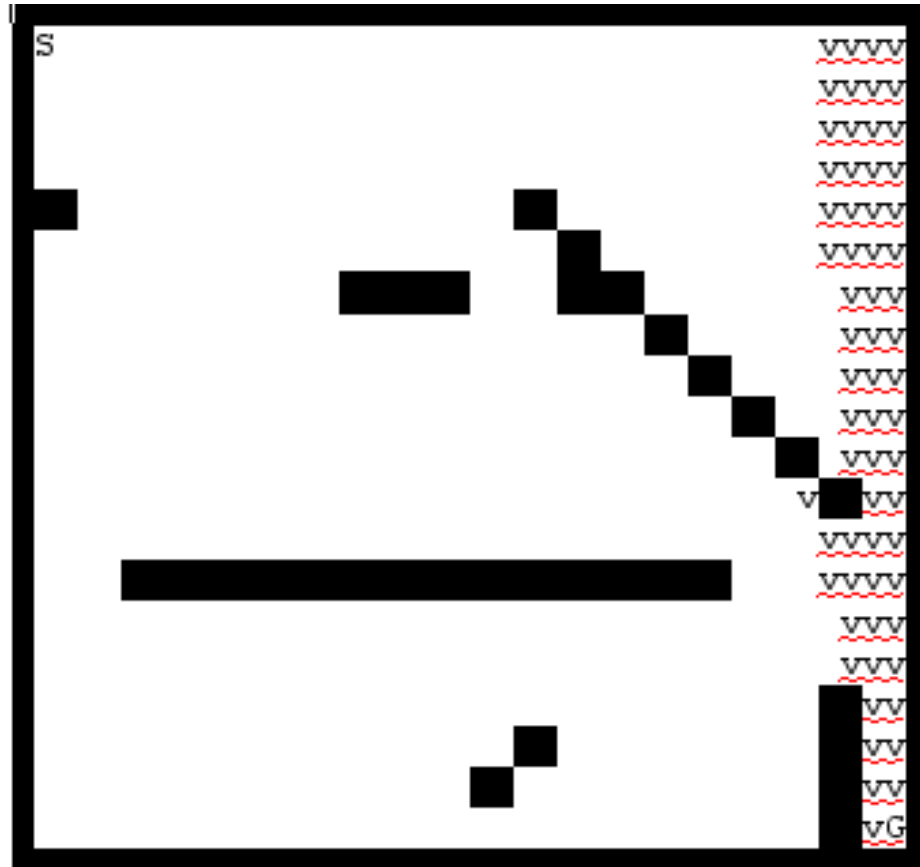
Figure 4.2: A PPP with 8% Goal visibility. 'v' symbols represent the squares from which the goal is visible to a perfect sensor.
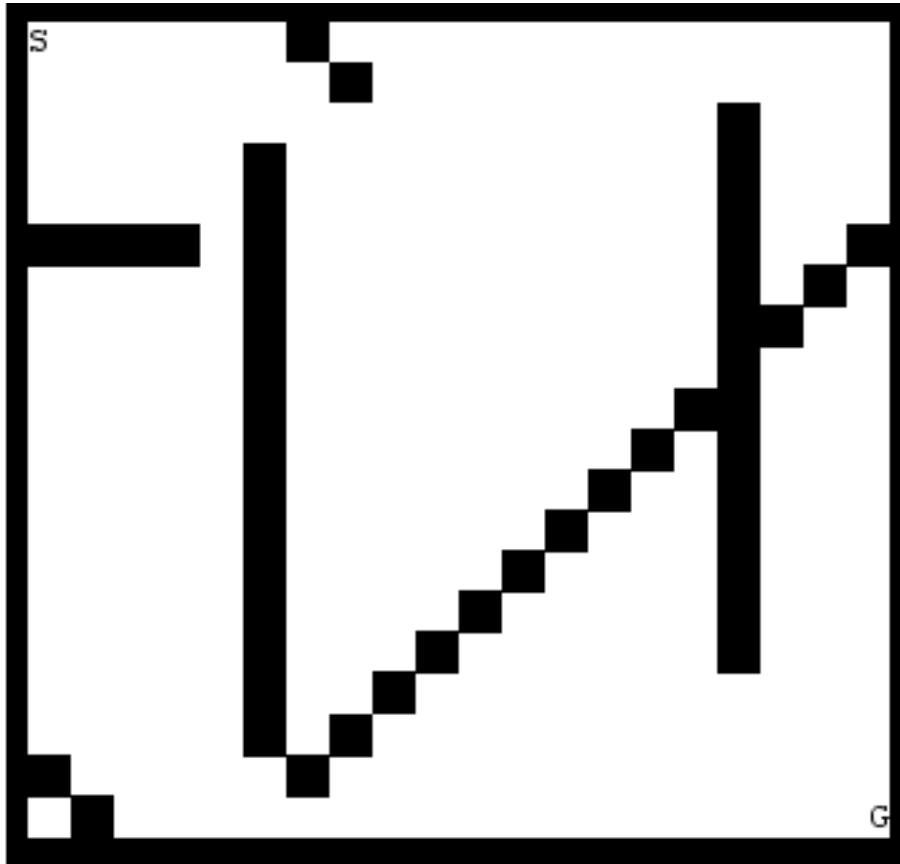
Figure 4.3: A PPP generated under Visibility Magnitude (Score=0.57).

be encouraged to spread the obstacles more evenly throughout the PPP. These five measurements would then be treated as a vector and the magnitude used as the final evaluation, acting as a measurement of the balance of visibilities from each position.

The fitness function was then tweaked to minimise this visibility; an example PPP produced under this function is provided in figure 4.3.

After prototyping a set of PPPs under this fitness function, I noted that many PPPs did indeed seem to spread obstacles more evenly across the environment. Furthermore, the additional viewpoints had forced the generator to make use of more of the budget of obstacles.

However, the generator only protects the start and goal positions - that is, it ensures these positions are never blocked by an obstacle. The other

three measurement positions were not under this protection; hence, a large fitness reward could be received by simply placing a single obstacle on each of these positions. In the worst case, this resulted in PPPs with obstacles clustered tightly around the start and goal corners, with single obstacles on the other three measurement positions. In addition, given that five measurements made up the vector, it was not intuitive to identify a difficult PPP given the magnitudes alone. Furthermore, it was not clear how to add measurements of other features of the PPP to this vector. As a result, I decided it was not a sufficient description of a PPP's difficulty.

### 4.7.3 Weighted Sum Function

In order to address the issue of incorporating measurements of many features of a PPP into a single evaluation, I reworked the visibility magnitude function into a sum of visibilities. This would allow additional feature measurements to be added as terms to this function; a more intuitive description than the vector. Furthermore, terms of the function could be weighted to influence the generator to focus more or less on each feature. As noted, the visibility measurements would make up one feature of this function.

A particular flaw of the previous measurements I had implemented was that a PPP could receive a good score while using few obstacles. This left the PPP environment wide open, resulting in high pass rates that were not reflective of the score given to them by the fitness function. I therefore introduced a cost to using few obstacles to the weighted sum, hoping to encourage the generator to use more of the budget of obstacles given to a PPP producing a less open environment. This term is described as a percentage of the maximum obstacles per PPP, which is a configurable generator parameter.

With this basic function in place, I continued prototyping PPPs and investigating to identify good and bad features. This led me, for example, to weighting the start and goal visibilities more highly such that the generator considered them more important than the other viewpoints. An example of a PPP generated with these weights is presented in figure 4.4.

In addition, I added new feature measurements to the function. I observed that many PPPs were generating with only one turn needed to reach the goal along the optimal path - this meant that the path was the same as an empty PPP. See figure 4.5 for an example of such a PPP. I therefore penalised PPPs with one turn in the fitness function.
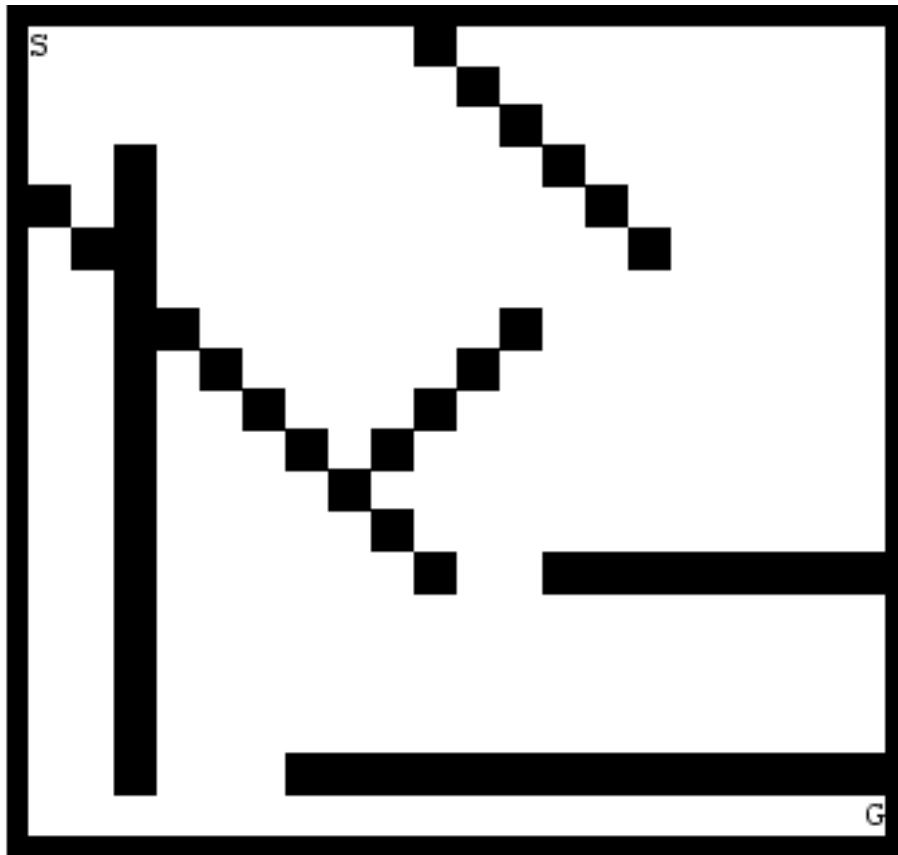
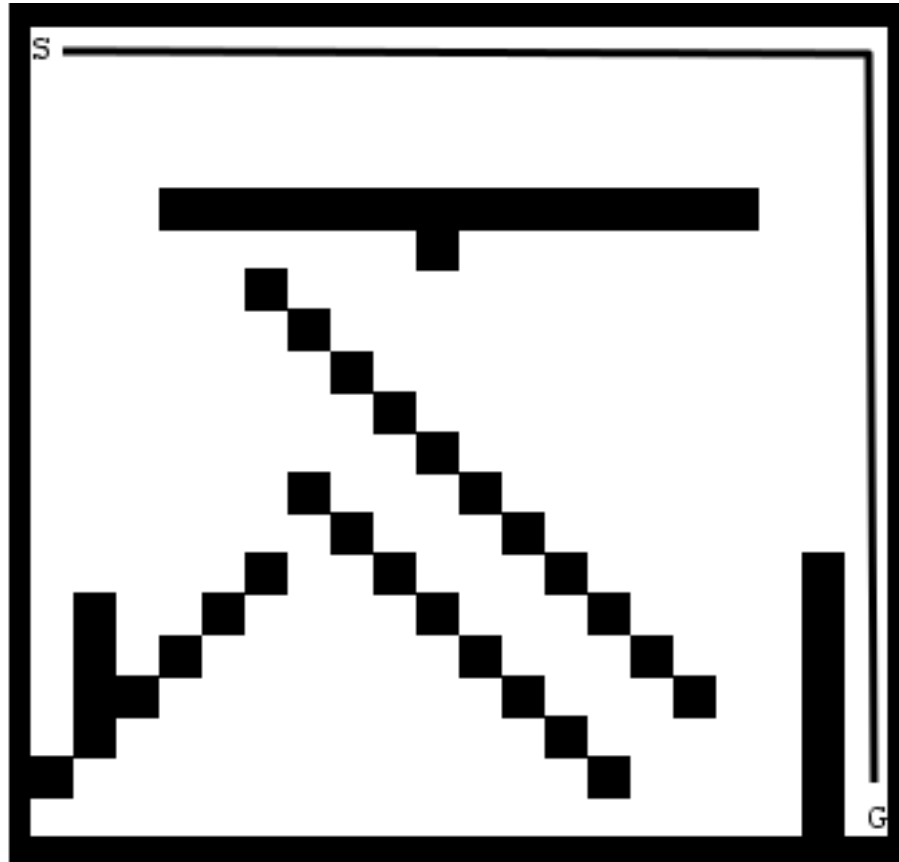Figure 4.4: A PPP with low start and goal position visibility.

Figure 4.5: A PPP with one turn on the optimal path, marked.

Furthermore, I noted that many PPPs with high pass rates across the agent set were often very open - obstacles were clustered up or usage was low, such that much of the environment was wide 'plains' rather than a more maze-like structure. I therefore measured the maximum openness horizontally and vertically across each row and column - that is, the maximum number of cells an agent could move in each without meeting an obstacle. I then averaged this across all rows and columns to produce a measurement of the average vertical and horizontal 'openness' of the PPP. Figure 4.6 shows an example PPP which is very open on both axes. I also experimented with other features, but removed them as the prototype results were worsened.
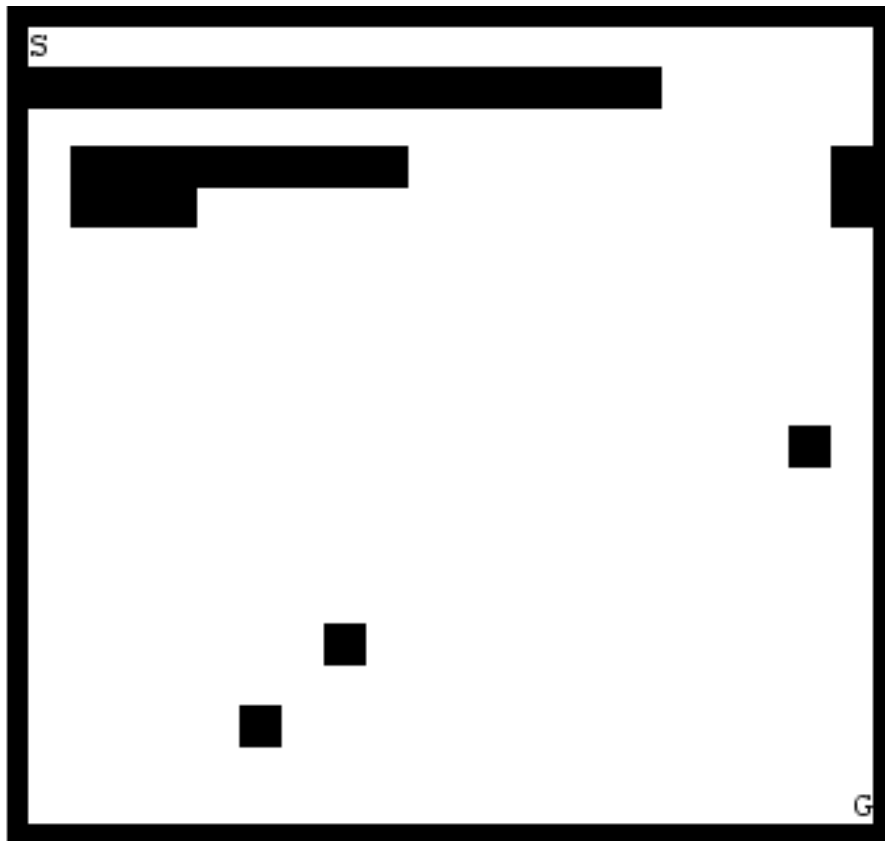
Figure 4.6: A very open PPP.

### 4.7.4 Final Evaluation Function

After the tweaks, weights and additional measurements had been added, the fitness function was finalised into the form given in equation 4.1. The generator seeks to minimise this function, a result of it being built initially upon the idea of minimising visibilities.

$$
\begin{aligned}
\sum (VisibilityFunction + 2 \times (1 - ObstacleUse) \\
+ 2 \times (AvgHorizonalOpenness) + 2 \times (AvgVerticalOpenness) \\
+ (2 \text{ if } OptimalTurns \equiv 1 \text{ else } 0)) \quad (4.1)
\end{aligned}
$$

Where 'VisibilityFunction' is the function given in equation 4.2, denoting the weighted sum of the number of cells visible from each position named:

$$
1.5 \times \text{Start} + \text{Centre} + \text{Top Right} + \text{Bottom Left} + 2.5 \times \text{Goal} \quad (4.2)
$$

## 4.8 Third Party Libraries

The following third party libraries were used during the project.

**D3.js 3.5.16 [16]**
A javascript library for manipulating documents based on data. This is used for visualisation of UPGMA tree structures, alongside a tree graph document produced by modifying an example graph.

**A\* Path planning [17]**
Pseudocode implementation of the A\* algorithm, slightly modified to make use of the simulator data structures.

**Line Drawing on Grids [18]**
Implementation of Linear Interpolation and Line drawing on grids used to produce lines of sight by the agent's sensor components. As above, slightly modified to make use of simulator data structures.

# 5 Evaluation

# Part III

# Closing Remarks

# 6 Conclusion

# 7 Future Work

# 8 Bibliography

[1] C. Umerson et al., "Autonomous Driving in Urban Environments: Boss and the Urban Challenge", J. Field Robotics Special Issue on the 2007 DARPA Urban Challenge, vol. 1 no. 8 pp. 425-466 Jun. 2008

[2] E. Guizzo. (2011, Oct. 18). How Googles Self Driving Car Works [Online]. Available: http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works,

[3] R. Alexander et al., "Situation coverage - a coverage criterion for testing autonomous robots", Dept. Computer Science, University of York, Report Number YCS-2015-496, Apr. 2015

[4] D. A. Ashlock et al., "Evolving A Diverse Collection of Robot Path Planning Problems" in IEEE Congress on Evolutionary Computation, Vancouver, BC 2006, pp. 1837 - 1844

[5] H. Wei, "Evolving path-planning problems to find faults in autonomous mobile robots", MSc thesis, Dept. Computer Science, University of York, York, N.Yorkshire, 2013

[6] H.Kitano, S. Tadokoro, "RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems", AI Magazine, Vol. 22 No. 1, pp. 39-52, Spring 2001

[7] A. Jacoff et al., "Performance Evaluation of Autonomous Mobile Robotics", Int. J. Industrial Robot, Vol. 29, No. 3, pp. 256-267, 2002

[8] D.A. Ashlock et al., emph"Search-based Procedural Generation of Maze-Like levels", IEEE. Trans. Comput. Intelligence and AI in Games, vol. 3 no. 3 pp 260-273 Sept. 2011

[9] Standard Test Mestods for Response Robots, ASTM International Standards Committee on Homeland Security Applications (E54.08.01), 2015

*8 Bibliography*

[10] J. Meyer, D. Filliat, "Map-based navigation in mobile robots:: II. A review of map-learning and path- planning strategies", Cognitive Systems Research, Vol. 4 no. 4, pp. 283-317 Dec. 2003

[11] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation" Computer, Vol. 22, no. 6, pp. 46âĂŞ57, Jun. 1989.

[12] S. Thrun, "Learning Occupancy Grid Maps with Forward Sensor Models", Autonomous Robots, Vol. 15 no.2, pp. 111âĂŞ127, Sept. 2003

[13] The Map Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot D. Lee Cambridge University Press, 1996

[14] H. Durrant-Whyte, T. Bailey, "Simultaneous Localization and Mapping: part 1", IEEE Robot. Automat. Mag., Vol. 13, No. 2, pp. 99-110, Jun. 2006

[15] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments", Proc. IEEE Robotics and Automation, San Diego, CA, May 1994

[16] M. Bostock D3.js 3.5.16, Available: https://d3js.org/

[17] A. Patel, A* Implementation Notes [Online] Available: http://theory.stanford.edu/ amitp/GameProgramming/ImplementationNotes.html

[18] Red Blob Games, Line Drawing on Grids [Online]. Available: http://www.redblobgames.com/grids/line-drawing.html

[19] K. Beck et al., (2001, Feb.) Agile Manifesto [Online]. Available: http://agilemanifesto.org/

[20] B. W. Boehm "A spiral model of software development and enhancement" Computer, Vol. 21, no. 5, pp. 61-72, May 1988.