Distributed Map Construction

Master Thesis Media & Knowledge Engineering

By

P.A.C. (Pjotr) Waasdorp, BSc.

Delft University of Technology

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

In

MEDIA & KNOWLEDGE ENGINEERING

By

Pjotr Waasdorp
Delft, the Netherlands

Delft University of Technology

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology,
The Netherlands
http://mmi.tudelft.nl

Distributed Map Construction

Date: 19 April 2010
Place: Delft

Author
Name: P.A.C. (Pjotr) Waasdorp
Student number: 1207482
Email: P.A.C.Waasdorp@student.tudelft.nl

Supervisor
Name: Prof. Dr. Drs. L.J.M. (Leon) Rothkrantz
Email: L.J.M.Rothkrantz@ewi.tudelft.nl

Abstract

Assume that two sub-graphs of a graph are available and that the graph represents the corridors in a building. Without using any previous knowledge on the entire graph, can these two sub-graphs be used to construct an extended view on the entire graph. To test this assumption a collection of algorithms will be designed and implemented which can fuse two sub-graphs (topological maps) together to achieve this extended view on the unknown environment. The main goal for this project will consist of the distributed construction of topological maps within a simulated environment, with the main focus on the used algorithm(s) for the fusion of multiple maps. This will be realized with the help of a "graph" data structure, a multi agent system and match and merge algorithms. In general the simulated environment will function as a test environment for the algorithms. A real life situation for the simulation environment could be, that it represents a building in which an emergent situation has occurred and in which individuals are exploring the unknown layout of the building in search for their personal goal(s). This system can be used by first responders who are exploring the building and are in search of casualties in case of a fire or a terrorist attack. This will all result in a simulation in which it is assumed that each explorer in the field is equipped with a Personal Digital Assistant (PDA) or a different handheld device and can communicate with the other explorers within its communication range, without the help of a centralized communication system. The explorer's traveled path is stored into their personal PDA and the software running on the PDA's shows the explorers their traveled path and if possible combines their map with the received map from the explorers they encounter. If a combination is possible the software on the explores PDA executes the algorithms to construct the most complete view on the environment as possible. The completed or partially completed map can also provide guidance to the explorers if this is needed.

Keywords: Multi Agent system, distributed, anonymous graphs, sub-graphs, map construction, wireless communication, exploration, Network

Thesis committee:

Chair:	Prof. Dr. Drs. L.J.M. Rothkrantz, Faculty EEMCS @ Delft Technical University
Committee member:	Dr. Ir. P. Wiggers, Faculty EEMCS @ Delft Technical University
Committee member:	Dr. Ir. Z. Yang, Faculty EEMCS @ Delft Technical University
Committee member:	Ir H. Geers, Faculty EEMCS @ Delft Technical University

Preface

This Master thesis will act as the final phase of the graduation project of the Master program Media & Knowledge Engineering which is a part of the Man Machine Interaction group of the faculty Electrical Engineering, Mathematics and Computer science at the Delft University of Technology.

This project is loosely based on a previous research assignment done by M. van Velden [1]; this research was part of the project “Crisis management using ad-hoc networks” at the DECIS lab. The project was a collaboration of Delft University of Technology, University of Amsterdam, THALES Netherlands and the Dutch organization for applied scientific research (TNO).

This current research however, does not have any affiliation with the fore mentioned project, businesses and university with the obvious exception of Delft University of Technology.

The project distinguishes itself by providing a stable simulation environment and a main focus on the distributed construction of topological maps with the help of different algorithms, and if time permits some additions to the new simulated world.

Acknowledgements

First of all I would like to express my gratitude to my supervisor Prof. Drs. Dr. L.J.M. (Leon) Rothkrantz, for giving me the opportunity to do this research assignment and guiding me through the process.
I would also like to thank my fellow students, with which I have experienced this period of my life, for all their input and companionship.

And last but not least my friends and family; Pieter, Trees, Sebastian, Sabah, Jeremy, Tirzah and my loving girlfriend Marloes, for always supporting me and having the confidence in me to complete this project successfully and with obtaining my Masters degree at Delft University of Technology.

Thanks, everybody!

Pjotr Waasdorp
Delft, the Netherlands

Contents

1 Introduction

1.1 Possible application

During a crisis situation such as an indoor fire or a partial building collapse there will almost always be trapped or injured people or important information which will have to be extracted from the building. In such a situation it is of vital importance that the rescue workers, who need to travel through the affected building, have a map of the internal layout of the specific building. Having such a map can help with finding goals e.g. casualties, fire extinguisher or exits and which can provide a current overview of the unknown environment.

Unfortunately such a map is not always available, up to date or doesn’t exists at all. And if such a map exist and is available, the internal layout of the building will most likely have changed after the crisis event has occurred. To solve these problems a new map of the new environment will have to be constructed in real time. However the construction of this new map can be a very time consuming and delicate job when done by one or by multiple individuals separately. This is time and effort which can better be spend on helping the casualties or saving the building and/or other important elements.

1.2 Proposed solution

Considering a building-like environment during a crisis situation, where individuals explore an unknown world. It is assumed that each individual is equipped with a PDA or an alternative handheld device which has wireless capabilities and can communicate with one another with the help of these capabilities. The software running on these PDA’s/handhelds keeps track of the traveled path and adds new information about the unknown world by communicating and exchanging information with other explorers. The users can also enter additional information about their location in the unknown environment, additional information such as blocked paths, blocked intersections or a specific property of the location. Because the users reside in a infrastructure less network they will communicate through a wireless Ad-hoc network.

Constructing a complete overview of a certain unknown environment with the use of limited knowledge only, can take a long time to achieve when using only one entity. To speed up the exploration process, multiple views on the unknown environment are needed to gather more information about this environment. A way to create multiple views on the unknown environment is to distribute the information gathering task to multiple entities in different parts of the unknown environment. The distribution of the exploration task provides more information in less time and effort, sharing and combining this information however is the next step in the process. Each entity will map its own traveled route and will make use of simulated wireless communication within a range. Within this range the information can be exchanged with other entities. Naturally the entity’s personal map has to have some added value to the other entities. This entails that the information has to be filtered on overlap and in some cases redundancy, after this filtering process the information will have to be merged with the information of the other entities that are within the communication range. Simply put, the entities will share their personal map with the other entities and by doing so they will eventually reach the complete map of the environment.

1.3 Communication

A possible application for this kind of algorithm(s) resides in the field of search & rescue. In this field the situation can occur in which information systems are not available. If such a situation occurs and there is no communications infrastructure at a certain area, it should still be possible to setup an infrastructure less network such as an ad-hoc network to establish communication with multiple parties. If communication can be established, sharing information is the next step in the process. Sharing information enables us to gather information concerning the dynamically changing state of the world and coordinate actions. All the gathered information can help with the effective mapping of the disaster area which in its turn can help with planning the resolution of the situation.

Ad-hoc networking technologies enable the exchange of information anywhere and anytime, without prior network infrastructure. When using handheld devices that operate in a wireless ad-hoc environment, it is still possible to communicate even when major infrastructural communication links have been damaged, destroyed or overloaded. So in case of a major disaster within a building or a city, emergency services can communicate without the need for preset-up access points or other such infrastructural requirements. In this thesis the assumption is made that the ad-hoc communication is available for all the users/explores in the area, which leaves the design and implementation of the multi-agent system which can operate in an environment without any previous knowledge of that environment, and the design and implementation of a system which can fuse location information from different users and distribute this information through the network.

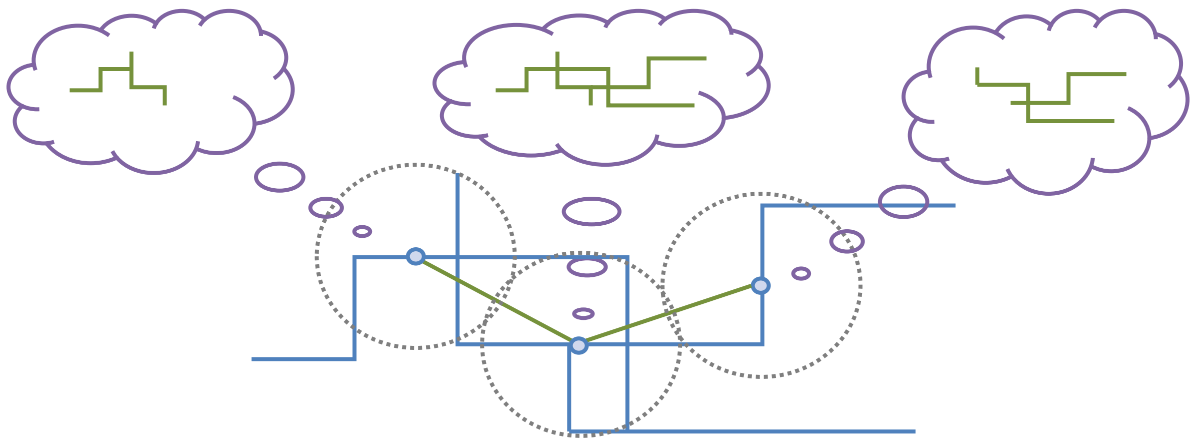


Figure 1: Communication between two explorers

1.4 Problem Definition

Assume that two sub-graphs of a graph are available and that the graph represents the corridors in a building. Without using any previous knowledge on the entire graph, can these two sub-graphs be used to construct an extended view on the entire graph.

Designing and implementing an algorithm or a collection of algorithms which can fuse two topological maps together to achieve this extend view on the unknown environment is the main goal for this research. Labeling cannot be used and the used graph is anonymous.

Related issues:

- The task of exploring the unknown environment will be distributed amongst multiple entities.
- The individual entities will have similar behavior and can use and detect location information.
- The environment has no pre-setup infrastructure and the entities in the environment do not have any previous knowledge of the world.
- A simulation will be created and with the help of this simulation the algorithm(s) can be examined and tested.
- The unknown world will be represented by a graph.
- The test results of the simulation will be examined and will provide us with information on the algorithm(s), in what way we can improve the algorithm(s) or in what way has the algorithm(s) failed its purpose.

How this research has been build up is displayed in Table 1.

Table 1: Research build-up

	Action	Goal
1	Literature study	Determining the level of research done in the field of graph construction and the used methods for solving the problem (state of the art).
2	Model / Design	Designing a method which solves the problem definition with the help of distributed exploration of the graph and without a central communication system.
3	Implementation	Implementing this design into a working simulation environment which can use a variation of maps and a diverse amount of exploring agent(s) within this simulation.
4	Tests	Testing the designed method with the simulation and comparing the results without the use of the method with the results when using the method.

1.5 Planned Solution Approach

To achieve a solution for this problem, the “Distributed Map Construction Application” (DMCA) will be developed. This application will provide a 2D simulation environment in which a graphical representation of a topological map will be constructed. This map will use a Graph data structure for constructing the map and providing paths for the entities in the graph. The simulation which the application provides will serve as a test environment for the match and merge algorithm(s).

The algorithm is the main focus of this research, and that is why a global description of the algorithm, in this stage will be given accompanied with an illustration in Figure 2. More details and information will be given in the following chapters.

Within the simulation two users are connected to each other through the wireless communication capabilities of their PDA’s. The communication can only occur between two PDA’s at a time. During the connection period both the personal maps are exchanged and will be analyzed and adapted by the algorithm(s) on the PDA’s.

Algorithm in very basic steps:

- Sort both the personal maps on edge count node tag and edge tag;
- Search for corresponding nodes (edge amount, edge length and rotation);
- If a corresponding node is found, expand and check the connected nodes and edges;
- Add it to the result list, and pick the best suitable one;
- Expand the personal map with the corresponding node as a reference point.

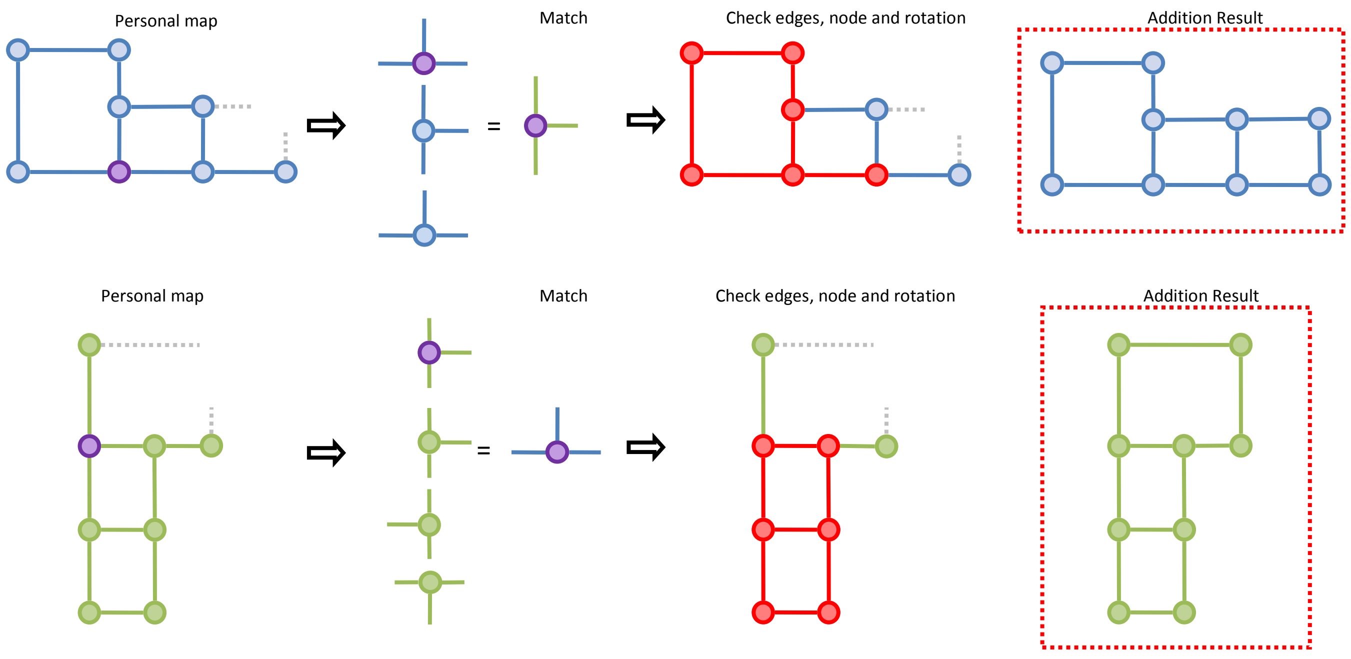


Figure 2: Match and merge

2 Background

Before actually working on the design of the algorithm and the implementation, additional research was required in order to get familiar with the concept of distributed map construction and all the preconditions it will entail. Gathering information about related research and projects was also be necessary to achieve a satisfying result so this will be covered in the next chapter. Exploring agents play a big part in this research that is why this chapter will cover a basic theory of agents. Further more in the following chapter a general description of some of the basic theoretical concepts and a short descriptions of related work will be displayed.

2.1 Software Agents

In a computer simulation of events and environments, the entity that will represent an inhabitant of that environment is called an agent. These agents are also known as so-called software agents. Using the term ‘agent’ is very convenient method to describe a complex software entity, which derives its actions and tasks from its own behavior and environment. A basic graphical representation of a simple reflex agent is shown in Figure 3.

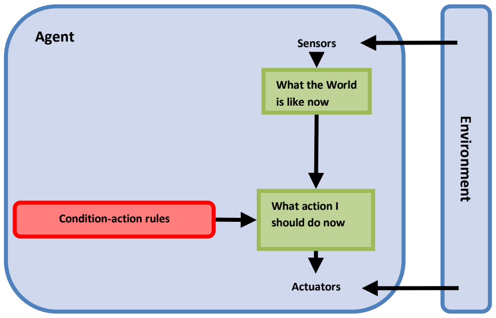


Figure 3: Agent representation

Agents can act with a certain amount of autonomy.
The definition of an agent includes more than one conceptual idea. These different conceptual ideas are mostly used to analyze the environment in which the agents operate. The concepts often relate to the way we naturally think about complex tasks. The design of an agent can also be approached from an implementation point of view, for example, an agent can have a persistence design, this means that the agent’s source code is run continuously and decides for itself when to perform some action. The agent can be autonomous; in that case, the agent is capable of task-selection, prioritization and goal-directed behavior without any human intervention. Furthermore an agent can have social abilities; with this ability they can engage other components or agents and coordinate or collaborate when performing a task. When an agent perceives the environment in which they operate and reacts upon the environment in an appropriate way, it is reactivity abilities.

There are many agent platforms available which (partly) implement the FIPA standards, some major platforms are: Agent Development Kit, April Agent Platform, FIPA-OS, Grasshopper, JACK Intelligent Agents, JADE, JAS, LEAP and ZEUS.

The ***D****istributed* ***M****ap* ***C****onstruction* ***A****pplication (DMCA)* contains a basic multi-agent system which communicates through the exchange of the agents personal travel data.

2.2 Agent Types

In [2] the concept of a software agent and the different types and application are discussed in great detail. However this is too much detail so to elaborate further on the differences between the concepts of software agents a subdivision of different types of agents will be given:

- Intelligent agent;
- Autonomous agent;
- Distributed agent;
- Multi agent system;
- Mobile agent;
- Fuzzy agent.

2.2.1 Intelligent agents

An intelligent agent is designed to adapt and/or to learn from a situation. The agent will adapt to a situation and will learn how to handle the situation the next time it comes around. The adaptation and the learning abilities both have a different approach to accomplice their tasks. The intelligent agent can adapt to a situation when it can sense its environment and adept according to the environment based on the choice between pre-defined problem solving algorithms and/or strategies. Learning abilities can come from simple trial and error but if this is the cause, the agent will have to have a notion of its goal and the level of success from its trail results. Learning can also be achieved with the use of a neural network. Using a neural network will mean that the agent will have to use examples to train itself and will have generalize the result as a desires outcome of the situation.

2.2.2 Autonomous agents

An autonomous agent is a software agent that, in theory, is autonomous. The agent is autonomous in the sense that it is self-containing and capable to make independent decisions so it can perform tasks or objectives to reach a goal. The agent is never really autonomous because it is always bound by rules and restrictions on which the agent architecture is based.

2.2.3 Distributed agents

The term distributed in this kind of agent is derived from the concept distributed computing. To simplify this method, distributed computing is a method to process different parts of a computer program on multiple computer systems. These computer systems communicate with each other through a network and complete the task by collaborating.
This method can also be applied with agents. The agents can be linked with each other and can for example share information and resources to produce a final result.

2.2.4 Multi agent systems (MAS)

A multi agent system or MAS is a system which consists of several software agents and collectively capable of reaching goals that are difficult to achieve by an individual agent.
Research with the help of a Multi Agent System is usually concerned with the development and the analysis of artificial intelligence problem solving. Some topics of research in MAS include:

- Beliefs, desires, and intentions (BDI),
- Cooperation and coordination,
- Organization,
- Communication,
- Negotiation,
- Distributed problem solving,
- Multi-agent learning,
- Scientific communities,
- Dependability and fault-tolerance.

In a multi agent system, multiple agents interact to achieve a common goal but can also achieve their goal(s) independently. This differs from a distributed agent because a distributed agent does not have the capability to achieve an objective by itself but has to communicate with others. If however an agent in the MAS does not have enough data, in other words a limited viewpoint, to achieve its goal(s) it will also have to collaborate with the other agents in the MAS.

Because some systems have a small amount of control, they are also referred to as a swarm system. A swarm system is populated with (simple) agents who interact with each other and with the environment to create a decentralized control structure known as global behavior or swarm intelligence [3]. An example of such a swarming system is an ant colony, where the shortest route to the goal and back to the colony is determined through information sharing between the ants. In this case, the shortest route is the one that has the strongest scent because it is most traveled.

MAS are very widely used for research topics. A lot of video games today are developed using MAS algorithms and MAS frameworks. A multi agent System can also be applied in transportation, logistics, graphics and many other fields. It is also widely used in networking and mobile technologies, to achieve automatic and dynamic load balancing and self-healing networks.

2.3 FIPA

The **F**oundation for **I**ntelligent **P**hysical **A**gents (FIPA) plays a major role in the world of agent technology. The FIPA is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents. The FIPA started its activities in 1995 with the aspiration to standardize different aspects related to agent technology and multi-agent systems.
FIPA’s model for agent systems is build around the concept of agent communication. Which means agents can pass semantically meaningful messages to one another in order to accomplish the tasks required by the application.

2.4 Graph theory

Graph theory is a term used in mathematics and computer science. The study of graphs is uses to model pair wise relations between objects from a collection of these objects. A "Graph" in the general context is a collection of vertices (or nodes) and a collection of edges. These edges connect pairs of vertices and represent the relationships with each of the vertices. In the context of computer science the graph is a data structure, to be more specific an abstract data type or ADT. A graph with vertices and edges is an ordered pair, and the scientific notation for this is: ***G: = (V, E)*** and is subject to the following conditions; **V** is a set of elements which are called vertices or nodes, and **E** is a multi set of unordered pairs of vertices called edges.

To help understand some of the terminology used within graph theory here are some descriptions. The vertices belonging to an edge are called **ends** or **end-vertices** of the edge. The **order** of a graph is | *V* | (the number of vertices). A graph's **size** is | *E* |, the number of edges. The **degree** of a vertex is the number of edges that connect to it, where an edge that connects to the vertex at both ends (a loop) is counted twice. The vertices connected to one vertex, are called **adjacent** to this vertex. In Figure 4 all the general used terms are visually represented.

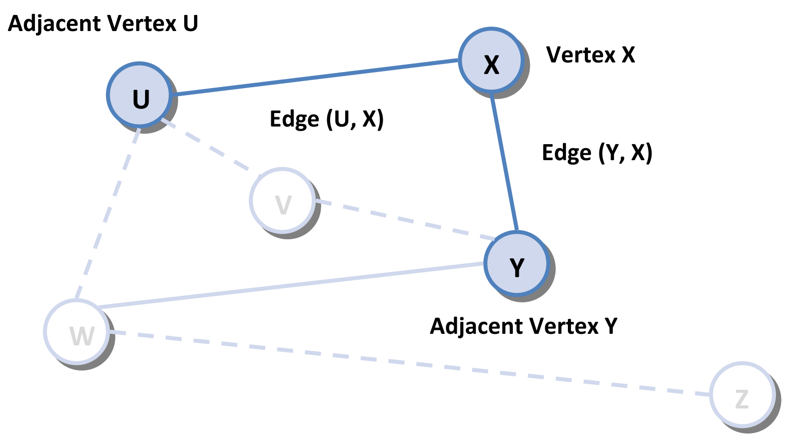


Figure 4: Visual graph representation

2.4.1 Directed & Undirected Graphs

A graph can be either *directed* or *undirected*:

- Directed: means that there is a distinction between the two connected vertices, one is outgoing and the other is incoming;
- Undirected: means there is no distinction between the two vertices associated with each edge.

Fout! Verwijzingsbron niet gevonden. and Figure 6 show the difference between an undirected graph and a directed graph.

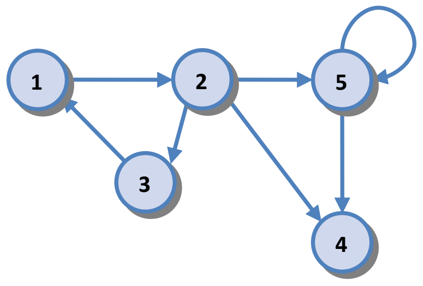


Figure 5: Directed graph

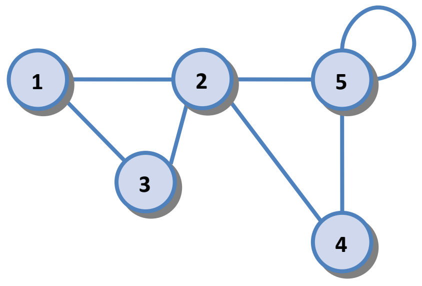


Figure 6: undirected graph

Within the graph theory there are even more different types of graphs or variations on the already shown graphs. However a detailed descriptions of these variations will not be given, they will only be named and accompanied with a short description.

- **Finite graph:** A finite graph is a graph with a finite amount of edges and vertices;
- **Simple graph:** A simple graph is an undirected graph and has only one edge between two different vertices. It can therefore have no self-loops;
- **Regular graph:** A regular graph is a graph where each vertex has the same number of adjacent vertices. Therefore every vertex has the same degree. A regular graph with vertices of degree *k* is called a *k*-regular graph or regular graph of degree *k*;
- **Weighted graph:** A weighted graph has a number (weight) assigned to each edge. The weights represent a value which is related to the problem. Weight of the total graph is the summation of all the weights;
- **Mixed graph:** A mixed graph is a graph in which some edges may be directed and some may be undirected;
- **Complete graph:** A graph is considered a complete graph when each pair of vertices has an edge connecting them;
- **Loop**: A loop is an edge (directed or undirected) which starts and ends on the same vertex.

2.4.2 Sub-graphs, induced sub-graphs

Finding a graph as a sub-graph of another graph is called a sub-graph isomorphism problem, and is a well known problem within the graph theory. Most graphs have the property to be hereditary for its sub-graphs, which basically means that a graph has a property if and only if all sub-graphs, or all induced sub-graphs have that property. Naturally this is very helpful but this poses a problem as well, because finding the maximal sub-graph in a given graph is called a clique problem and is NP-complete. The same thing goes for induced sub-graphs (also NP-complete).

2.4.3 Graph traversal

As stated before if our used graph was a labeled graph traversal could be used to explore the unknown graph. The general meaning of graph traversal is visiting every node in the graph (and maybe processing the values of the visited node). Graph traversal works in two specific procedures, Depth First Search or Breadth First Search. For both methods (Depth- & breadth first search) the starting node can be an arbitrary random node in the graph, the next node to be chosen depends on which method is used. The depth first search and the breadth first search will be clarified a little more in the next two paragraphs.

2.4.4 Depth first search

When using the depth first search the next node which will be visited will be the adjacent node which has the highest depth value. The next node will be the one with the next highest depth value. This process will repeat itself until all the adjacent node for that node are visited, and do this for every visited node. This is illustrated in Figure 7 and accompanied with a small description. If V1 is the starting point of the traversal, and as you can see V2, V3 and V8 are its adjacent vertices. All three of these nodes must be visited but when using the depth first search, the first node to visit would be V8 and after that the V2 and V3 will be visited.

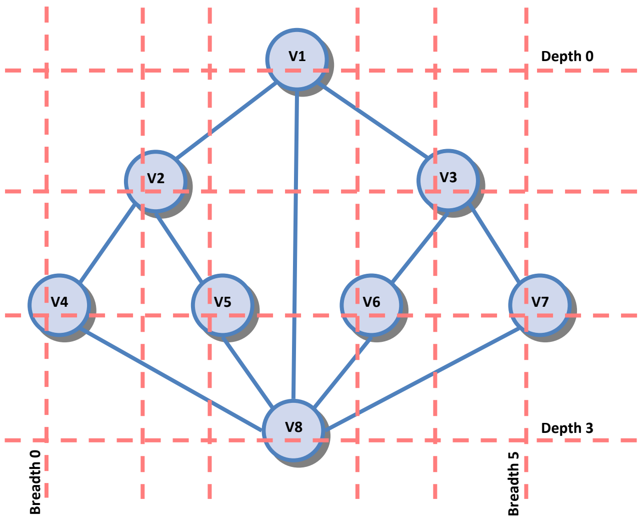


Figure 7: Breadth & depth

2.4.5 Breadth first search

In general the breadth first search is the same search algorithm as the depth first search with the exception that it does not chooses the next node with the highest depth value but with the least breadth value instead.

If V1 is the starting point of the traversal, and as you can see V2, V3 and V8 are its adjacent vertices. All three of these nodes must be visited but when using the breadth first search, the first node to visit would be V2 and after that the V8 and after that V3 will be visited.

2.4.6 Applications of graph theory

Graph theory is usually applied in combination with a labeled graphs or alternatives on a labeled graph. However the structure of a graph is so comprehensive that many different problems can be represented by a graph.

These representations can vary from a simple visual representation of an application workflow to a complex routing algorithm. Other fields where a graph is often used are biology, chemistry, physics, computer-chip design and even in sociology (for measurements in social networking software). Because of this diversity of the graph implementation, the development of algorithms to handle graphs is of major interest in computer science.
In computer science the graph is more often used for analysis, for example to analyze traffic. But there are different categories in which the graph analysis is used:

- Analysis to determine structural properties of network (for example the distribution of vertex degrees and the diameter of the graph) ;
- Analysis to find a measureable quantity within the network (for example the level of vehicular flow within any portion of a transportation network);
- Analysis of dynamical properties of networks.

A lot of graph implementations already exit but the production of useful ones in various domains remain an active area of research [4].

2.5 Location information

Gathering environmental information about the surroundings can help with identifying a certain location. Temperature, colors, light and noise are just some of the examples of information sources from which a location can be identified. Environment information is a part of the physical context information of this location and could help with the identification of a location.

Although the location of a certain area is of great importance in this project this kind of environmental information is beyond the scope of this project and is there for not an issue.

If the **G**lobal **P**ositioning **S**ystem (explained in the next paragraph) or a similar system is available the location of each node can easily be verified, however the GPS does not work properly indoors and the possibility that the GPS is not available in an emergent situation is more likely. When the GPS is not available the location of the node or nodes will have to be determined with different methods. And distributing this task and sharing information is the proposed method to use in this project.

2.6 Global Positioning System (GPS)

Global positioning system or GPS is a technology which uses satellite data to provide highly accurate location determination and this is used a lot in navigation- and information systems. GPS was developed by the United States Department of Defense but a European effort to create its own GPS called ***Galileo*** is also being developed and will act as an alternative and a complementary system to the American GPS. The biggest difference between the two is that Galileo is intended to be more precise down to the meter range.

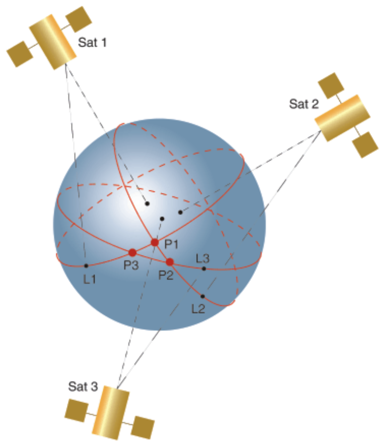


Figure 8: GPS satellites

The GPS system consists of between 24 and 32 medium earth orbit satellites. These satellites circle the earth in a controlled orbit and transmit accurate microwaves signals to a GPS receiver which in its turn can determine its location, traveling speed, traveling direction and time. The GPS device determines these values by calculating its distance from each visible satellite. The best result will be obtained if the GPS device has at least three visible satellites, when this occurs the device can use triangulation to calculate latitude and longitude coordinates (Figure 8). If a fourth satellite is in the right position the GPS device can also calculate the altitude. Altitude is the elevation above mean sea level; with this the height of the location can be determined.

A GPS system could make this research unnecessary but the environment is purposed to be indoors where a GPS system does not function. And the focus is placed on the collaboration between multiple virtual agents to achieve a complete view on the unknown environment.

3 Related work

Exploring a graph like world and mapping the unknown environment has been studied extensively due to its multitude of applications in a variety of different areas. Usually these studies concentrate on constructing the map of the unknown world with the help of a single entity. This entity will usually be a robot or a virtual representation of something similar, like a software agent. The unknown world in which the entity is exploring is usually represented as a two dimensional graph.

Such a graph consists of nodes and edges. Within a single entity implementation the entity will have to travel trough each edge and pass each node to have the full view on the explored world in the form of a map. As mentioned, most of the related work on exploration of unknown graphs has been limited to a single agent and usually in a labeled graph. In labeled graphs each node is uniquely identifiable, and this will always result in a possible exploration and mapping of the graph by traversing it.

3.1 Exploration using a single agent

The majority of research in the field of graph exploration has been done with the use of single agent exploration. That is why this area of research will be focused on first.

3.1.1 Right- Hand-on-the-Wall

The right-hand-on-the-wall method is an interesting approach of the exploration problem. In the research paper [5] they consider the problem of *perpetual traversal* by a single agent in an anonymous undirected graph *G*. In this research they set the requirements to:

1) A deterministic algorithm;
2) Each node is visited within O(n) moves;
3) The agent uses no memory, it can only use the label of the current node;
4) No marking of the underlying graph is allowed;
5) No additional information is stored in the graph (e.g. routing tables, spanning tree) except the ability to distinguish between the incident edges (called *Local Orientation*).

This problem is unsolvable, as been proven in [6] and [7] even for much less restrictive setting. Their approach is to somewhat relax the requirement (5).

Their adjusted approach related to the manner in which the agent explorers the graph. They fixed the following traversal algorithm: *"Start by taking the edge with the smallest label. Afterwards, whenever you come to a node, continue by taking the successor edge (in the local orientation) to the edge via which you arrived"* and ask whether it is for every undirected graph possible to assign the local orientations in such a way that the resulting perpetual traversal visits every node in $O(n)$ moves.

This leads to an extremely simple, memoryless, yet efficient traversal algorithm divided in three steps.

1. Constructing a (directed) subgraph H;
    a. The undirected graph induced by the bidirectional edges of H is connected and contains all vertices of G;
    b. Each vertex *v* of ***H*** is either saturated or has exactly two unidirectional underlying edges, one incoming and one outgoing;
    c. ***H*** contains $O(n)$ edges.
2. Constructing witness cycle C.
    a. For each vertex of ***H*** define a witness ordering. (These orderings define one or more cycles in ***H***.);
3. Locally modify the orderings in some nodes in order to merge these cycles into one super-cycle ***C*** containing all vertices, while maintaining Right-Hand-traversability.

They showed that for every connected simple undirected graph the local orientations in the vertices can be chosen in a way that creates a right-hand rule cyclical walk of length at most $10n$ covering all vertices. They have also showed how to maintain this property even when more vertices and edges are added to the graph.

3.1.2 Labeling

In many cases and/or studies on labeled graph exploration, the emphasize is focused on the minimized cost of exploration, in terms of the number of moves (edge traversals) or memory usage by the agent [8]and [9]. Most of these studies however do not require a construction of the explored graph. In contradiction to our research which is more focused on the construction of the explored graph and not the minimal memory usages or minimal edge traversal.

Most of the related work tried to solve the exploration of an unknown graph by allowing the exploring agents to mark the nodes in a certain manner [5]. In labeled graphs, each node is uniquely identifiable and hence, it is always possible to explore and map the graph simply by traversing it.

Marking or labeling the nodes has been done in several different approaches, for example in [10] a method is proposed which labels a node by dropping a pebble on the node to mark it.
Another method is provided in [11], they used a set of distinct markers for the exploration of unlabeled and undirected graphs. These alternatives of labeling or marking a graph are basically the same approach of solving the exploration problem.

A different way of marking a node is used in [12]. Use two cooperative agents and position one of these agents on a node and let the other explore new edges. However this is not a very effective use of your second agent and diminishes the solution to a single agent implementation.

3.1.3 Tree exploration

When the unknown graph is not labeled, another method will have to be used to reconstruct it. in [8] it is shown that it is possible to construct the unknown graph when it has a tree design.
A mobile agent has to explorer an undirected graph by visiting all its nodes and traversing all edges, without any prior knowledge of the topology of the graph nor of its size. The task of visiting all the nodes of a network is fundamental in searching for data stored at unknown nodes of the network, and traversing all edges is often required in network maintenance and when looking for defective components.

If nodes and edges have unique labels, this can easily be done by depth-first-search. However, in some navigation problems in unknown environments such unique labeling may not be available, or limited sensory capabilities may prevent from perceiving such labels. Hence it is important to be able to explore anonymous graphs without unique labeling of nodes or edges. Unfortunately, arbitrary graphs cannot be explored under such weak assumptions. As witnessed in the case of cycle.

Without any labels or the possibility of putting marks, it is clearly impossible to explorer a cycle of unknown size and stop. If labeling or marking is not available the class of graphs that can be explored has to be restricted to connected graphs without cycle, i.e. trees.

In this paper the problem of graph exploration is studied under very weak assumptions:

- Not labeling or marking allowed (focus on tree exploration);
- The agent or robot can distinguish edges at a node (otherwise it couldn't even explore the 3 leaved star);
- All edges are locally labeled;
- No coherence between those local labels is assumed;
- Minimal amount of memory as possible.

The following tasks are considered:

- Exploration with stop: starting at any node of the tree, the agent has to traverse all edges and has to stop at some nodes;
- Exploration with return: starting at any node of the tree, the agent has to traverse all edges and has to stop at the starting node.

In both cases the goals is to find an algorithm which uses the least amount of memory.

This resulted in the fact that the exploration with stop requires more memory because it needs this to decide when to stop in the anonymous environment. The algorithm used for exploration with return is much more memory-efficient than a basic DFS[1]-algorithm and performs better in general.

[1] Depth-first-search

3.2 Exploration using multiple agents

Research on exploration with multiple agents in an unlabeled graph is not as common as the single agent implementation.

3.2.1 Map construction of unknown graphs by multiple agents

In [13] a distributed version of the graph exploration and mapping problem is considered. Where a mobile agent has to traverse the edges of an unlabelled (i.e., anonymous) graph and return to its starting point, building a map of the graph in the process.

In this case, instead of using a single agent, there are k identical (i.e., mutually indistinguishable) agents initially dispersed among the n nodes of the graph. The agents can communicate by writing to the small public bulletin boards available at each node. The objective is for each agent to build an identically labeled map of the graph; we call this the Labeled Map Construction problem (LMC problem).

This problem is much more difficult than exploration by a single agent, because it involves achieving cooperation among multiple agents. In fact, this problem is deterministically unsolvable in some cases. In this research deterministic algorithms are represented that successfully and efficiently solve the problem under the condition that the values of n and k are co-prime[2] with each other.

Their solution works in two stages:

1. In the first stage, the agents independently explore and mark the graph, each obtaining a partial map of the graph.
2. In the second stage, the partial maps are combined to construct a map of the whole graph.

Their algorithm solves the Labeled Map construction problem, while also performing leader election among the agents and constructing a spanning tree of the graph, under the assumption that the size of the graph, n is co-prime with the number of agents, k. Under this assumption, the problem is always deterministically solvable irrespective of the structure of the graph G or the initial placement of the agents on it.

An important property of their solutions is that the agents explicitly terminates their execution even in the unsolvable cases. For termination detection, the value of at least one of n or k must be known to the agents. They showed that when the value of both these parameters are known, it is possible to terminate earlier, thus improving the complexity of the solution.

[2] two integers *a* and *b* are said to be co-prime if they have no common positive factor other than 1 or, if their greatest common divisor is 1.

3.2.2 Improved Distributed Exploration of Anonymous Networks

In many research papers the problem of constructing a labeled map of an anonymous and asynchronous network is addressed. An algorithm that explores and maps the network by using *k* identical agents that have no prior knowledge of the network topology, a lot like the research in this paper. The algorithm provided by [13] for mapping the network requires that *n* and *k* are co-prime. The improved algorithm, presented in [14], requires at most $O(m \cdot \log k)$ edge traversals, while it uses $O(m \cdot k)$ edge traversals (m is the number of edges in the network). The size of the whiteboard memory needed in the algorithm is the same as the amount used in [13], $O(\log n)$. Introducing a modification to resolve issues of electing first "local leaders" among adjacent candidates, which otherwise may deadlock the process.

The LMC[3] algorithm used in this research proceeds in phases. The traversal algorithm is invoked by the agents several times in each phase. It moves an agent from one node to another, based on the agent's label. All agents' labels are initially identical but are later refined to distinguish agents from one another. Nodes and edges visited by an agent are marked with the agent's label (unless already marked by the same or larger label). An agent *territory* is the sub graph of G consisting of the nodes and edges marked by the agent.

By using phases they can bound the number of graph traversals through node v to $O(\log k)$. In each phase, an agent searches to find another agent in the same phase so that they can merge and become one agent in a higher phase. As they go to higher phases, fewer agents remain (there are at most $k \cdot 2^{-P+1}$ agents in phase $p>0$). The search is performed by agents, that annex nodes by writing the label of the agent to the node's whiteboard.

The traversal algorithm used here, is the Depth-First-Search algorithm intended for the family of undirected networks. It is possible to generalize the results to other families of networks by using other traversal algorithms. As showed here for Depth-First-Search - the fact that two identical agents may visit the same node does not cause the traversal to get "confused". If the algorithm is used for other families of graphs by using other traversals, care must be taken to make sure these traversals do not get "confused" when the routes of identical agents collide.

The algorithm can be used, with some modifications, to follow the ideas of [15] and solve some other possible cases. The presented algorithm constructs a map only when it can detect a successful termination since the map construction is included in the leader procedure. It is possible to have every agent construct a map as it goes. This will however increase the bit complexity. However, it will ensure that a map is constructed by the last remaining agent also in the case that n and k are co-prime, but neither n nor k is known.

[3] Labeled Map Construction

3.2.3 Map Construction and Exploration by Mobile Agents Scattered in a Network

In [16] the map construction problem is considered in a simple, connected graph by a set of mobile computation entities or agents that start from scattered locations throughout the graph. The problem is further complicated by dangerous elements, nodes and links, in the graph that eliminate agents traversing or arriving at them.

The agents working in the graph communicate using a limited amount of storage at each node and work asynchronously. In this research paper a deterministic algorithm is used that solves the exploration and map construction problems. The end result is also a rooted spanning tree and the election of a leader. The total cost of the algorithm is $O(n_s\, m)$ total number of moves, where m is the number of links in the network and n_s is the number of safe nodes, improving the existing $O(m^2)$ bound.

The problem, called “*Dangerous Graph Exploration”* (DGE) is for a team of agents to explore the network and, within finite time, generate a map of the safe part with an indication at each safe site of the dangerous ports (black links or leading to a black hole). Solving this problem is a dangerous (and possibly impossible) task for the agents. The goal is for at least one agent from the team to survive, with all surviving agents terminating with the map.

Within finite time, after at most $O(nm)$ moves, a rooted spanning tree of G_S will be constructed, all safe edges will be indicated as such, all ports in G_S leading to a black hole or to a black edge will be marked as dangerous.

3.2.4 Distributed Exploration of Anonymous Graphs by Multiple Agents

The problem of exploration and mapping of an unknown environment modeled as a graph, by multiple identical mobile agents that are dispersed among the nodes of the graph [17]. The objective is for each agent to build an identically labeled map of the graph; we call this the Labeled Map Construction problem. The problem is of much practical importance because having such a map facilitates the navigation and coordination among the agents when they are lost in an unknown environment. A deterministic solutions to the problem under the weakest possible setting, assuming the agents only have the knowledge of either the size of the graph or the number of agents present.

In this research a protocol is presented that will allow a team of k anonymous agents scattered in an unknown unlabelled graph of n nodes and m links to construct a map of the graph, and elect a leader among the agents, using no more than O(m.k) edge traversals. They then show that the complexity of this algorithm can be improved to O(mlog k), when both n and k are known a-priori to the agents (or at least the value of gcd(n, k) is known along with either n or k).

Distibuted traversal

The agents will explore the graph collectively, in such a way that the total number of edge traversals is minimized. Each agent can traverse an area around its homebase(starting point), while avoiding the parts being explored by the other agents. During the exploration, the agent needs to remember the path to its homebase, so that it does not get lost. Each agent stores in its memory the sequence of labels (in order) of edges traversed by it, starting from the homebase. This way the agent can build a partial map of the territory that it marks.

Merging maps: Spanning tree construction

To obtain the map of the whole graph, the maps constructed by the agents, after the exploration algorithm need to be merged. The task of merging together the maps of the agents, is complicated by the fact that the maps constructed by two agents may look exactly similar. Also there may be cyclic 'NT' edges connecting two nodes of the same tree. To avoid the cyclic edges, we would first construct a spanning tree of the graph by joining the trees marked by different agents.

The merging algorithm uses the exploration algorithm as a procedure. The algorithm proceeds in phases, where in each phase, some agents become passive i.e. they stop participating in the algorithm. Agents communicate by writing certain symbols on the whiteboards. Two special symbols would be used which are called the 'ADD-ME' symbol and the 'DEFEATED' symbol. An agent can be in one of three states: **Active**, **Defeated** or **Passive**. Each agent is active at the time it starts the algorithm, but it may become defeated and subsequently passive, during some phase of the algorithm. When an agent becomes passive during a phase, it keeps waiting at its current location till the end of the algorithm. At the time an agent starts the algorithm, it knows the value of either n or k.

Phase 0: Every agent has finished the exploration of its territory and contains three values nl.:

- Ph = Phase number (initially set to 1);
- Nc = Node count (the number of nodes in its territory);
- Ac = Agent count (the number of agents in its territory, initially set to 1).

Phase 1: If an agent is active it will execute the following steps:

- Agent will perform a depth-first traversal of its territory using the map; recall that a territory is a tree. During the traversal the agent writes its Token on the whiteboard6 of each node in its tree;
- During this step, the agent compares its token with the tokens in adjacent trees;
- After the comparison of Tokens, the agent takes one of the following actions;
    - [<] If the token is larger it writes an 'ADD-ME' symbol on the whiteboard of node v and returns to node u. Node u is the terinal node and the edge e is the bridge edge. The agent does a complete traversal of the territory writing 'Defeated' symbols on each node in its territory. It's a defeated agent;
    - [=] If the token is equal, it ignores the token and return to its own tree and continues with its traversal;

- [>] If the token is smaller it waits at node v until it finds a 'Defeated' symbol. It then goes back to node u and continues with the traversal.

Defeated agents continue with the traversal and token comparison, and will never write an 'ADD-ME' symbol. After traversal completion the agent returns to the terminal node and marks the bridge edge as the junction point in its own map. At this stage, the agent becomes passive and does not participate in the algorithm anymore. If an 'ADD-ME' symbol is encountered, the agent will delete the symbol and wait at that node until the agent which had labeled the node, returns to the node. The waiting agent will adopt the labeling agents data and sets the current node as the place where it acquired this new information (the acquisition point).

The last step in phase 1 is to update the tokens. If the agent does not becoming passive, it extends its territory and updates its Token, before starting the next phase. The agent adds together the Node-count and Agent-count values respectively, from all the acquired Tokens, including its own Token, to get the new values of Node-count Nc, and Agent-count Ac. The agent also constructs a new map by merging the acquired maps with its own map. The resulting map constructed by the agent defines its new territory. If the agent finds that the new node-count is equal to n (or the agent-count is equal to k), then it reaches the termination condition. Otherwise, it proceeds with the next phase.

Final phase: Complete-Map

1. The leader agent executes a depth-first traversal of the spanning tree, writing node labels on the appropriate whiteboards.
2. The leader agent traverses the graph, adding the non-tree edges to the map.
3. The leader agent traverses the spanning tree to communicate the full map to all the agents.

The problem of constructing a labeled map of an unknown unlabelled graph by a team of identical asynchronous mobile agents initially dispersed among the nodes of the graph. Multiple agents can collectively explore the whole graph, each obtaining a partial map of the graph. To combine the partial maps and construct a single combined map of the whole graph, there has to be some agreement between the agents.

The provided algorithm in this research, achieves this agreement and elects one of the agents as a leader, under the necessary condition that gcd(n, k) = 1, where n is the size of the graph and k is the number of agents. The algorithm only works if the value of one of n or k is known to the agents. However, if the value of gcd(n, k) is also known to the agents then it possible to have a more efficient algorithm.

3.3 Leader election

Leader election is the process of designating a single process as the organizer of some task distributed among several nodes. Before the task has begun, all network nodes are unaware of which node will serve as the "leader," or coordinator, of the task. After the leader election algorithm has been executed, each node throughout the network recognizes a particular, unique node as the task leader.

The network nodes communicate among themselves in order to decide which of them will get into the "leader" state. For the election of leader a method is needed in order to break the symmetry among them. For example, if each node has unique identities, then the nodes can compare their identities, and decide that the node with the highest identity is the leader. Leader election algorithms are designed to be economical in terms of total bytes transmitted, and time.

3.4 Rendezvous dilemma

The rendezvous dilemma is related to the prisoner's dilemma and a simplified scenario can be formulated like this:

“Two people have a date in a park they have never been to before. Arriving separately in the park, they are both surprised to discover that it is a huge area and consequently they cannot find one another. In this situation each person has to choose between waiting in a fixed place in the hope that the other will find them, or else starting to look for the other in the hope that they have chosen to wait somewhere.”

If both choose to wait, they will never meet. But if both choose to walk there is a change that they will meet. If one chooses to wait and the other chooses to walk there is a theoretical certainty that they will meet eventually. In practice this would need an infinite amount of time for it to be guaranteed. The question posed is: What strategies should they choose to maximize their probability of meeting each other?

The rendezvous problem as shown here is a problem of theoretical interest but it also has real-world problems with applications in a variety of fields. One of these applications is the map construction of an unknown environment, with the help of mobile agents. When dealing with mobile agent models the rendezvous problem has been studied extensively.

The rendezvous problem in these settings basically entails that all the mobile agents gather in the same node and creates the graph from this point on. Creating a result for this problem often entails the use of probabilistic algorithms. If a deterministic solution to the problem is preferred Yu and Yung in [18] investigated the use of synchronous graphs of known topology and in [19] the rendezvous problem on synchronous rings and graphs with distinct labels is studied.

The rendezvous problem in the context of anonymous agents in an unlabelled ring network is also studied in [20] , [21] and [22] also studied the rendezvous problem within a graph with a sense of direction. In [22] the rendezvous problem was challenged with the assumption that the edge labeling on the graph provides a sense of direction.

3.5 ManetLoc

ManetLoc is a location based approach to distributed world knowledge in mobile ad-hoc networks [23] [1]. The algorithm which ManetLoc uses is the inspiration for the algorithm used in the DMCA. To fully understand some of the choices made in favor of the algorithm in the DMCA we will first provide some details on the ManetLoc project algorithm.

The algorithm first step is to detect a closed loop in the graph. This condition is based on the idea that if a user would travel long enough in a building he will eventually always return to a location that was visited before from a different direction. If a closed loop is detected correctly it provides very valuable information, because it is a requirement for the consistency of the map, and a consistent map is a requirement for the match and merge algorithm in the ManetLoc project. The closed loop detection is explained with the use of a simple square shaped graph, illustrated in Figure 9.

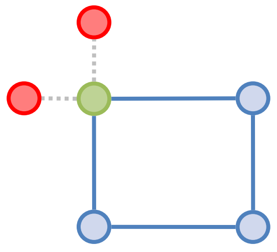


Figure 9: Closing the loop

The two red nodes in the graph are viewed by the system as the same node (the green node). Before the loop can be closed the ManetLoc tries to build multiple hypotheses concerning possible loops. To do this ManetLoc roughly follows a procedure that checks for the existence of vertexes next to a given one that might close a certain loop. If two matching nodes can be found a hypothesis is started.

To construct a consistent graph, two matching vertexes will have to be found, small adjustments can be made to the endpoints of edges if this is necessary. Whenever a loop hypothesis has been formed it can start testing it by comparing edge lengths of the supposed loop with new measurements. These measurements will result in accepting or rejecting the loop.

The matching of two maps is split up into three separate parts e.g.:

- Node matching;
- Growing hypotheses;
- Combining hypotheses.

3.5.1 Node matching

The node matching procedure is building a list of all nodes that match each other in the two maps. Two nodes match if they have the same edge directions. A match will also be accepted if the node needs a rotation to match. Fuzzy variables for node attributes are not used in the simulation but in a real environment or an extension of the simulation these variables might occur.

3.5.2 Hypothesis growth

The second step in the matching process is that of the hypothesis growth. As soon as the list of nodes is available the matches will be tested on corresponding pairs of edges and the paired vertices. If the edges are compatible and the nodes at the ends are also compatible they are added to the hypothesis. If the edges or nodes are incompatible the entire hypothesis is rejected. The vertices are tested with the same type of criteria and similarity tests used to form the initial pair. Edges may also have both exactly and inexactly known attributes. The accepted hypotheses are the unique matches surviving the growing process. Duplicate hypotheses are avoided by keeping a table of node pairs. When nodes are paired during the growth phase, the corresponding entry is marked in the table. This entry corresponds to an initial pairing of nodes. A sub graph in one map can be matched to multiple sub graphs in the other under separate hypotheses, but a pair of matched nodes with a given edge correspondence can appear in only one hypothesis. The matching and growing process is repeated until all valid vertex pairings are examined.

3.5.3 Multiple hypotheses

If the hypothesis is successful, the hypothesis growing process described above results in a list of possibly multiple hypotheses within the same rotation. From these hypotheses one has to be selected. Before this takes place it is possible that if such a list contains more than one hypothesis, some of these entries are consistent with each other. These hypotheses are then combined with each other into one larger hypothesis cluster. After the system has chosen a hypothesis cluster, the next step is to merge (or flatten) the two maps into one single map. Estimates of path lengths can be updated by combining the measurements from the two maps for corresponding edges. The edge orientations at the corresponding vertices can be similarly merged.

The parts of a given map that are missing should be added. The merging process is globally performed in four steps:

- Rotate the received map so its orientation matches the local node’s map;
- Shift the rotated map so its coordinates match the local agent’s map;
- Add any new vertexes from the rotated and shifted map to the local node’s map;
- Connect everything together (update edge lengths, check for inconsistency’s etc.).

4 Model and algorithm design

In this chapter the key elements of the algorithm(s) are reviewed and explained. Together with the following chapter “Architecture” this will provide a deeper understanding on how and why the algorithm(s) are implemented in the way they are. The overall design issues like: requirements, constraints and purposes will also be elaborated on. Furthermore the model and algorithm design chapter will provide a better view on all the aspects of the design and shows the data structures which were used. The designs of the algorithms will be accompanied by some pseudo code.

4.1 Requirements & Constraints

When designing any system, there are requirements and constraints that need to be taken into account. For these requirements and constraints a distinction is made between requirements and constraints for a real world implementation and for a simulation. In this case the algorithms are all executed within a simulation but are meant to be used on a handheld device in the real world. The handheld device however does not have the same performance as a modern desktop computer upon which this application is developed and tested. A requirement could be that the computation and execution of the algorithm are within the limits of the handheld device performance. Another issue is that if the algorithm(s) would be implemented into a real world situation, the environment issues will have be taken into account as well. An example could be that the application will switch to speech or provide more and brighter colors in its display when the visibility of the environment is minimal.

However these requirements are purely based on a real world implementation which is not the focus of this project. In contrast with the following constraints, the focus will only be on the Algorithm(s), the simulation environment, the design issues of the simulation and all the different parts of the DMCA.

Requirements for a real world implementation could be:

- The algorithms should be able to run on a handheld device;
- Data traffic has to be kept to an absolute minimum;
- A node should have ways of providing local info on its location and its edges to the agent; i.e. user input, GPS if available and the pedometer(to record traveled distance) for increased accuracy.

Constraints for this project:

- Only a simulation will be implemented for the thesis;
- It will be assumed everything takes place in a rectilinear labyrinth- like environment, with horizontal and vertical edge directions;
- All the edges must be connected to a minimum of two nodes;
- In a square world, opposite edges have equal length;
- Only nodes can function as intersections of two edges;
- An agent can only communicate with one other agent at a time;
- The algorithm can’t make use of location data such as an x and y-value of node locations.

4.2 Algorithms

The main focus of this research is on the match and merges algorithms. To test and execute these algorithms a simulation environment was created with the development of the **D**istributed **M**ap **C**onstruction **A**pplication or DMCA. The design and implementation of the DMCA will be elaborated on in coming chapters, and in the following paragraphs the basic algorithms on which this project is constructed will be discussed.

4.2.1 Exploration algorithm

The exploration algorithm (the name is very self explanatory) is used by the virtual agents, to explore the map. The virtual agent starts at a random node in the map and receives one out of four traveling speeds; the variation in traveling speed has been implemented to minimize the chance that two users/agents have the same traveling speed and are constantly exploring together and sharing only with each other. In the starting node all the connected edges are unknown so the agent picks a random edge to start its journey. The agent will keep track of every node and edge it visits in its personal map.

Each time the virtual agent reaches a new node it can immediately see all the possible directions it can choose from, this entails that all the possible edges are known as well. The reason why the virtual agent has this ability is because in real life situation, if a person is standing at a crossroad he too can see every direction in which he/she can travel. If the virtual has explored the entire edge only then will the edge be tagged as visited, this counts for the last visited node and the target node of the edge. If the virtual agent reaches a node and the node has multiple unvisited edges it will choose a random edge from the unvisited edges. If the virtual agent encounters a node with only one unvisited edge it will always choose the unvisited node.

```
If(node){
Node n = new Node();
List l = new List();
Edge ed = new Edge();

n = RetrievePersonalPoint();
        foreach(Edge e in n.EdgeList){
                if(e.visited == false){
                l.Add(e);
                }
        }
ed = getRandomEdge(l);
updateEdgelistCurrent node(ed, node);
updateEdgelistConnectednode(ed, node);
}
Return ed;
```

Figure 11: Pseudo code exploration algorithm

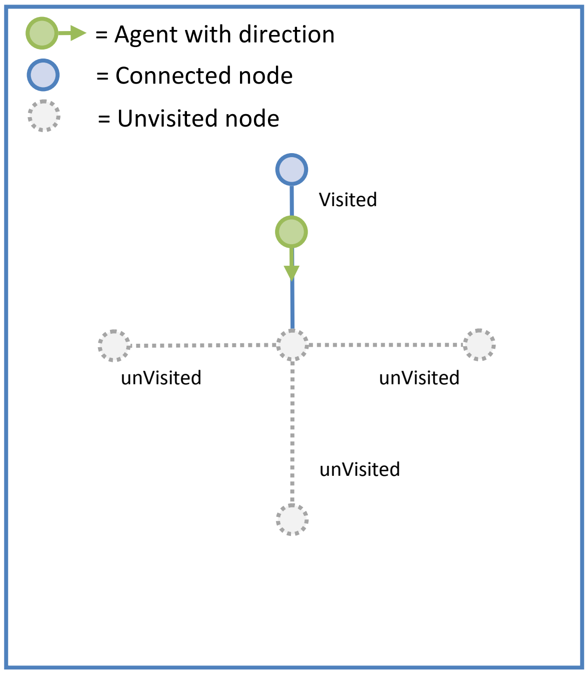


Figure 10: Node with visited & unvisited edges

However if the virtual agent reaches a node with only visited edges (let's call this a complete node) it will encounter a problem with the next routing choice. The problem with the random decision when reaching a complete node is that there is no certainty that the agent will choose an edge which leads to an incomplete node is very small. This kind of situation is illustrated in Figure 12. If the agent is backed into a corner it is more likely that it will choose the 'wrong' edge (illustrated by the red edges) to continue its route.

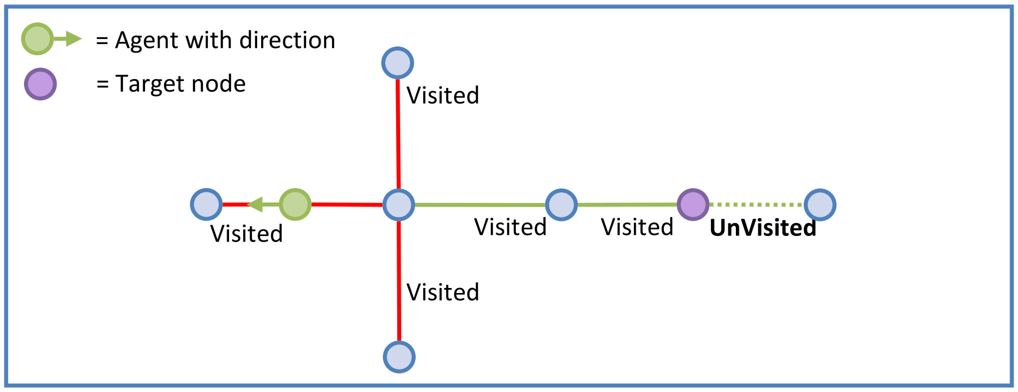


Figure 12: Random algorithm problem

That is why a search algorithm will have to be implemented, this algorithm will search for the nearest incomplete node (illustrated by the purple node in Figure 12) and provides the route to the correct node.

The search algorithm for the nearest incomplete node is basically a level counter it starts off at the source node and expands its search from the source node(s) to all the adjacent nodes. Each time the algorithm expands to a new level of adjacent nodes it will add one (+1) to the level value. The connected edge from the original source node with the lowest level value contains the nearest incomplete node.

Because of the dynamic environment the agent is in the algorithms will trigger every time the agent encounters a complete node without a direct route to an incomplete node. When the algorithm is triggered, the agent will have to choose a connected edge which will provide the route to the nearest incomplete node. Every edge will therefore be examined for the available route. From each connected edge the connected node is examined first, if this is complete the source node and the connected node will be added to the 'visited nodes' list and the level will be set to the value 1. The next step is to gather the adjacent nodes from the connected node which are not in the 'visited nodes' list.

If there aren't any nodes available, this route is impossible and will be discarded. If there are new nodes available these will also be examined for completeness and if this is true this step will be iterated and with each step the level will be increased by a value of one (+1).

This process will result in a level value for every edge; the edge with the lowest value will be chosen to continue the agents' path. If multiple edges have the same level value a random edge from these edges will be chosen.

The illustration in Figure 13 shows the level counting for one connected edge from the green source node.

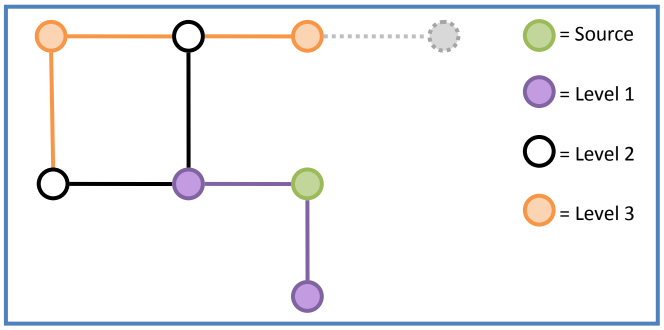


Figure 13: Level counting

4.2.2 Matching algorithm

The matching algorithm plays a major part in the project; together with the expanding algorithm it is actually the main algorithm which was developed. The matching algorithm will try to find a reference point in the received personal map so it can map the received map onto its own personal map, and by doing so extend its own map with additional information.

The algorithm is triggered when two virtual agents are within communication range from each other and the two virtual agents have not communicated with each other for a period of time. Communication can only occur between two agents at a time, so if agent A is communicating with agent B both A and B are unavailable for other virtual agents. One of the virtual agents is always the instigator of the communication, this virtual agent will receives the designation of current agent and the connected agent will receive the designation of connected agent.

When the situation of communication occurs, both virtual agents first check if their personal maps are complete[4] or not. If the personal map is complete, the virtual agent with the complete map will not try to find a reference node or expand its own personal map because it's already has the entire map. Trying to find a reference node or expanding will be useless because it will not add any more information. For the other virtual agent this is very beneficial because if a reference point can be found it will instantly have the entire map. If however, the receiving agent has an incomplete personal map it will trigger the share algorithm (see sharing algorithm in paragraph 5.2.4) and if this returns a negative response the personal map of one or both involved virtual agents are exchanged and the connected agent is added to the share list.

[4] A map is complete if it does not contain any nodes with unvisited edges and if the target node on the current edge is visited.

For a more detailed explanation of the algorithm it is assumed that the received map is incomplete (has nodes with unvisited edges). After the received map is tested on completeness both the received- and the own personal map will be sorted. The sorting has three elements:

- Node completeness;
- The amount of connected edges;
- Node and edge tag.

The first step when sorting all the nodes is to filter both the personal maps for complete nodes[5]. All the incomplete nodes are discarded and the sorting will continue with the remaining complete nodes.

Next the nodes will be sorted on the amount of connected edges, so all the nodes with the highest amount of connected edges (in this case four) will be added first, after that the nodes with three connected edges, then the nodes with two connected edges and last the nodes with one connected edge.

The third and last sort property is the node and edge tag. All the nodes and edges have a tag which represents a characteristic of that node or edge and if the tag is set to null the node or edge has no specific characteristic, but if the tag has a different value it has a specific characteristic.

The highest ranking node will have a node tag and one or multiple connected edges will also have an edge tag. Next in line are the nodes with just a node tag, after that the nodes with one or multiple edge tag, and last are the nodes without node- and edge tags. Note that this is done for all the nodes with the same edge amount. The sorting result is illustrated in Figure 14. After the sorting the matching algorithm will use the sorted maps in the sorted order.

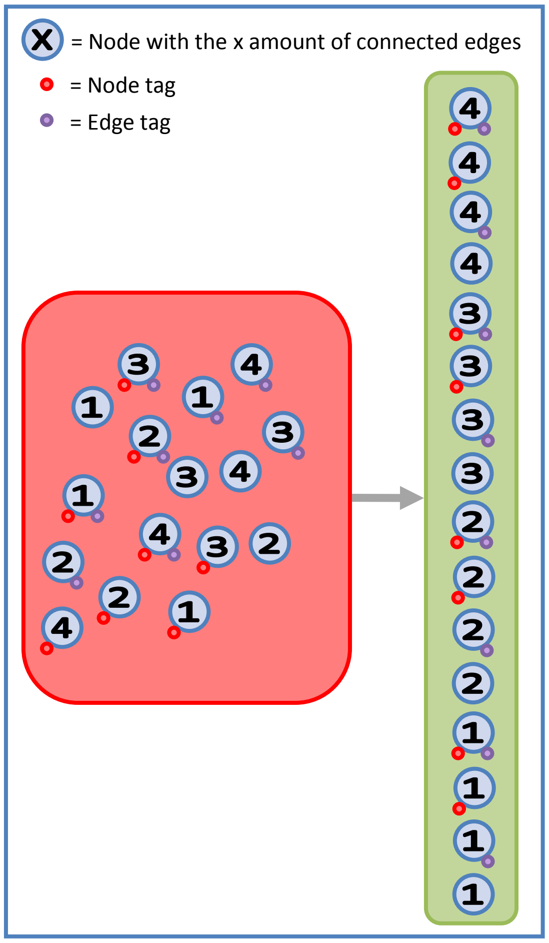


Figure 14: Sorting results

[5] A complete node is a node without any unvisited edges

4.2.3 Reference Node

The next step is to compare the nodes from the own personal map with the nodes from the received personal map. If a node passes this initial test it is added to a temporary list which will be examined further in a later stadium.
The initial check will consist of:

- A check for similar amount of connected edges;
- A check for the rotation of the node in comparison with the current agent node;
- A check for similar edge lengths;
- A check for similar node tag;
- A check for similar edge tag;
- A check for the connected nodes and edges which are available will be checked on the same properties.

The first check is very basic, it only compares the edge count of the two nodes and if the edge count is equal it will go to the next check, but if it fails it's not a match and the combination will be discarded. A deviation in this process occurs when a node has two connected edges. A two edged node can have two shapes in this case; these two shapes are illustrated in Figure 15. These two types are also checked when checking the edge count.

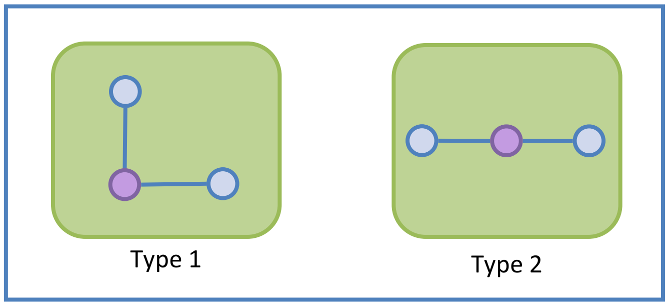


Figure 15: The two types of nodes with two edges

The rotation of the node in comparison with the current agent node is basically the check if the node is rotated or not and if it is rotated how many degrees it is rotated. Figure 16 illustrates the rotation options. The rotation value will be stored in a temporary variable which we can use in a later stadium in the algorithm.

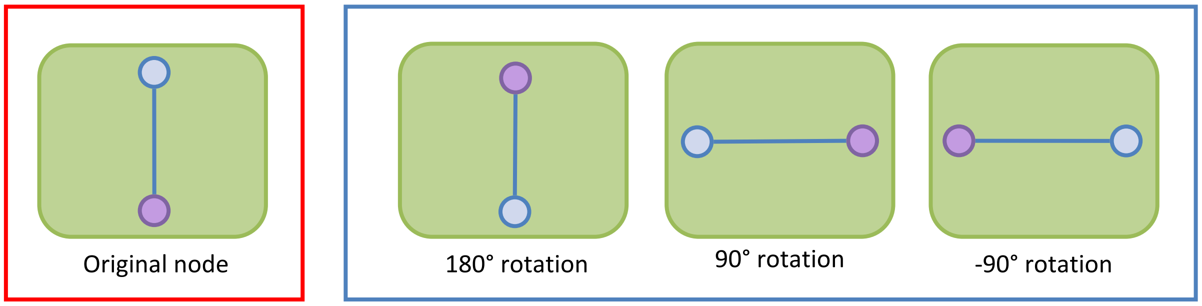


Figure 16: Node rotation options

The rotation value is only determined in the first initial check. When we expand the similarity check to the connected nodes the initial acquired rotation value is used in the continuation of the algorithm. Next the node tag will be checked, the nodes and the edges both have tags which can vary in its denotation. But if the node tags are the same the algorithm will continue to the next step. In this implementation only two kinds of tags are used a neutral tag and the blocked tag. In the implementation these tags are represented by integer values, neutral = 0 and blocked = 1. In the future it is possible to add more kinds of tags to indicate a location property such as the presents of a staircase or a fire extinguisher, these additions could have a very positive effect on the algorithm because it adds more detailed information on the specific location. Next step in the initial similarity check is to compare the edge length of all the connected edges to the current node. This test is a difficult element in this test because the edge length is easily accessible in the implementation but in a real world scenario the length of an edge is not that easy to estimate or to compare. In this implementation we use the true edge length with a deviation of 5% because the algorithm will not change in its design when we use estimated value with the exception that the algorithm will have less positive results when working with estimated values. After the edge lengths have been checked the edge tags will be checked. The tag properties are the same as the node tags and are also implemented as integer values. If the edge tags are also similar the initial similarity test is achieved with success and the next major step in the algorithm will begin. To provide a general look into the matching algorithm, the pseudo code is displayed in Figure 17.

```
ArrayList OwnSortedMap =  SortMap(OnwMap);
ArrayList ReceivedSortedMap = SortMap(ReceivedMap);
ArrayList SimilarNodes = new ArrayList();

Foreach(Node n in OwnSortedMap){
        SimilarNodes = getSimilarNodes(ReceivedSortedMap)
        Foreach(Node sn in SimilarNodes){
                If(compareNode(n,sn)){
                        If(amount < minAmount){
                                If(nextLevel(n,sn)){
                                Return true;
                                }
                        }
                }
                Else{
                continue;
                }
        }
}
Return false;
```

Figure 17: Pseudo code matching algorithm

4.2.4 Matching algorithm flowchart

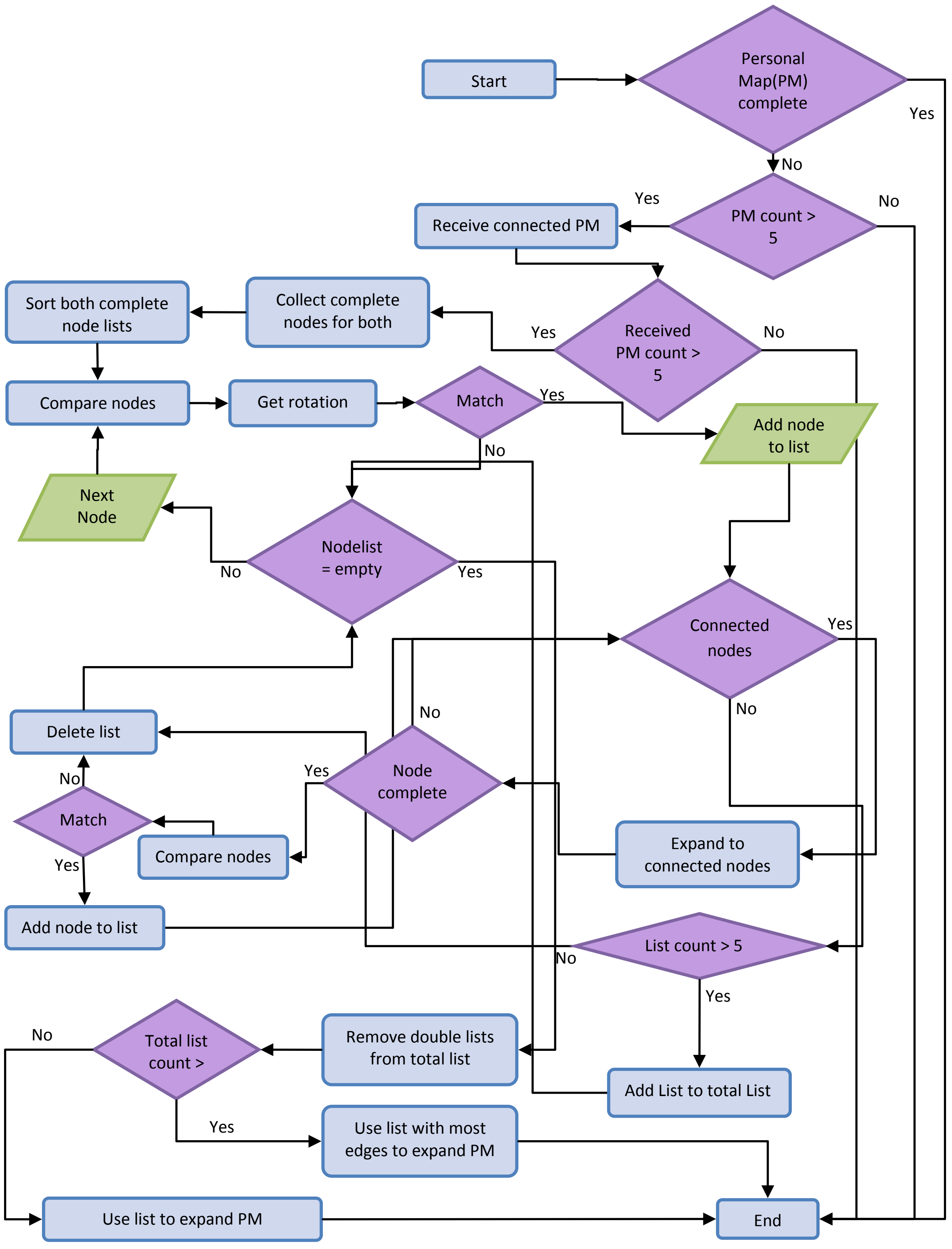


Figure 18: Matching algorithm flowchart

4.2.5 Expanding algorithm

Now that an acceptable node has been found, or in most cases multiple acceptable nodes, have been found. The next step is to check if the acceptable node can be expanded with new connected nodes and these nodes with their connected nodes. An iteration process will be executed and will be continued until an incomplete node is encountered or if a node is rejected. Every time an acceptable node is encountered the node is added to a possibility list and if a node is rejected the entire possibility list is discarded.

A connected node is accepted when the same elements as the initial similarity check are accepted:

- The edge count;
- Node tag;
- Rotation value;
- Edge tag;
- Edge length.

The rotation value for all the connected nodes are compared with the value of the starting node, it will have to have the same value as that node. If this is not the case, the segments do not match and this attempt will be abandoned. If multiple acceptable nodes are found in the initial similarity test, a possibility list for each similar node is constructed.

For a possibility list (a segment of the graph) to be accepted a minimum of four nodes in total (this includes the starting node) will have to be accepted. The amount of four accepted nodes is not totally arbitrary; on the contrary it is a well contemplated number. It is based on the maximum amount of connected edges to a node and as seen in the ManetLoc [1] [23] the same amount is used to check the consistency of the graph. The only difference is that the ManetLoc uses a loop of four nodes in a loop instead of four connected nodes in an arbitrary sequence.

There is a good change that multiple possibility lists have four or more accepted nodes. If this is the case the possibility list with the most accepted nodes (the longest list) will be selected as the most probable one. A problem here is that four node with four edges each, contain much more information than four nodes each with only one edge. That is why if this situation occurs the multiple lists will be checked on edge count. The one with the highest edge count will be selected as the most probable and this segment will be used to merge the two personal maps together.

4.2.6 Merging algorithm

When a matching segment has been found we will have to merge the existing personal map from the current agent with a new segment of the connected agent. The matching segment is used to verify the node in both graphs as “the same node” and with this node it is possible to add new information to the graph. Merging the two elements to one new personal map is there for expanding the existing personal map of the current agent. This algorithm is similar to the algorithm used in the ManetLoc project [1] [23] for the merger of two maps.

Starting off with the algorithm the collected reference node is used rotate the received personal map based on the rotation value. The next step is to translate the received personal map to the coordinates of the local personal map and place the two maps on top of each other. Following this procedure is the adding of new information such as new nodes and edges to the existing personal map or updating existing nodes and edges with new information.

The agent can check if a node is already present in a graph because the corresponding node is mapped on the personal map and with the edge length we can check the coordinates of the potential new node. If a node already exists at these coordinates the node will not be added and if at these coordinates there is no node present it will be added.

The merging process is approximately performed in four steps:

- Rotate the received map so its orientation matches the current nodes’ map;
- Translate the received map so its coordinates match the local agents’ map;
- Add new nodes from the rotated and shifted map to the current nodes’ map;
- Connect everything together (update edges, update map).

```
rotatedMap = rotateMap(rotation, receivedMap);
translatedReceivedMap = transLateMap(rotatedMap, referenceNode);
ArrayList newMap = new ArrayList();
newMap = PersonalMap;

foreach(Node n in PersonalMap){
        if(containsNode(translatedReceivedMap, n)){
                newMap.Add(n);
                adjustEdges(n,newMap);
        }
}
Return newMap;
```

Figure 19: Pseudo code merging algorithm

4.2.7 Flowchart merging algorithm

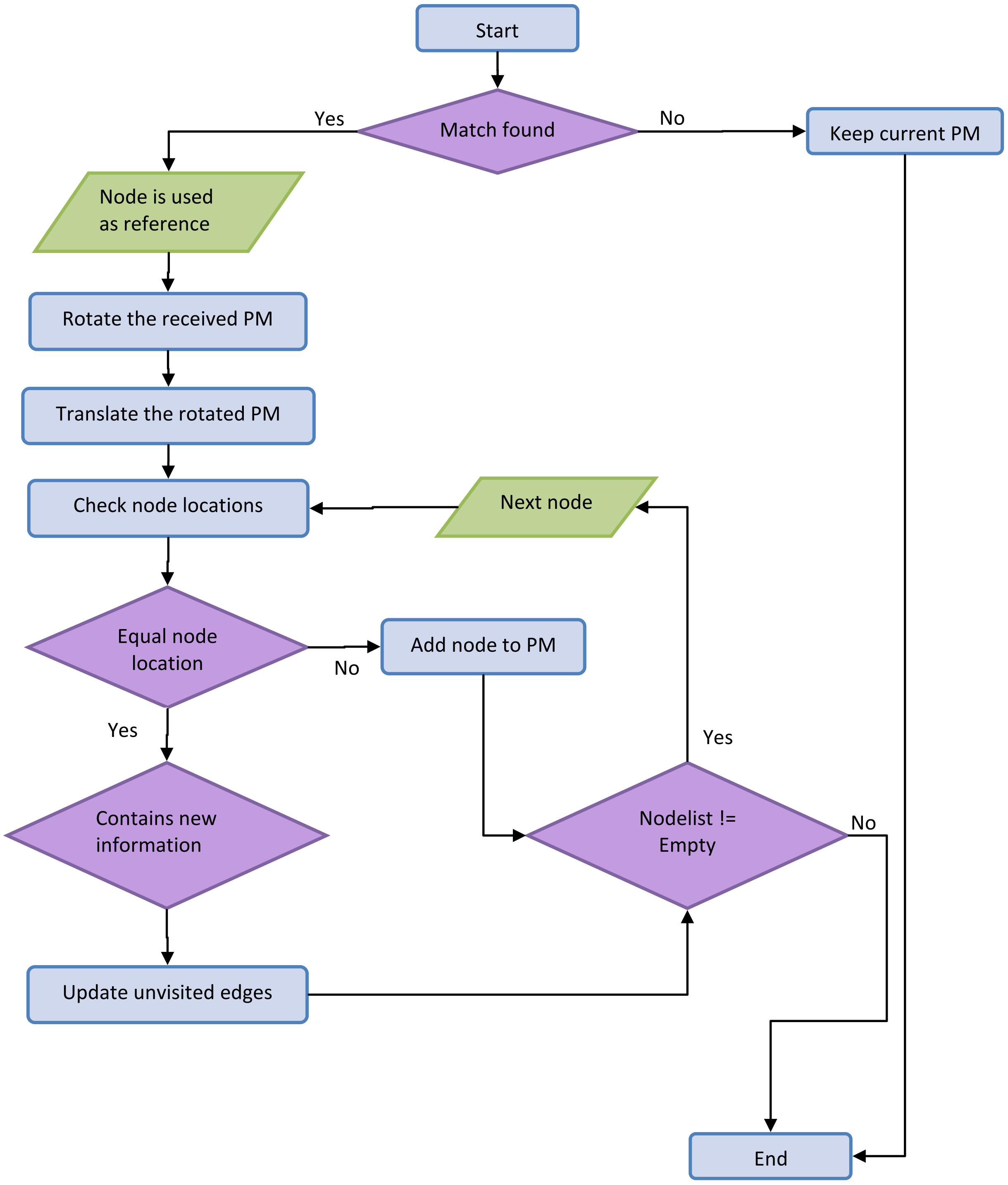


Figure 20: Merging algorithm flowchart

4.2.8 Sharing algorithm

The sharing algorithm is a relatively simple part of the simulation but very useful. The algorithm is executed on each PDA, and when two users connect and exchange their personal maps it adds the users it connects with to its sharing list. The user entry will stay in this list for a set amount of seconds, in this case three seconds, after which the user will be removed from the list.

When a user is still in the agents sharing list it may not share its information with that agent, this measure prevents two agents to share their personal maps within a short amount of time and creates a window of opportunity for other agents to connect with the current agent.

The sharing algorithm is illustrated in Figure 21.

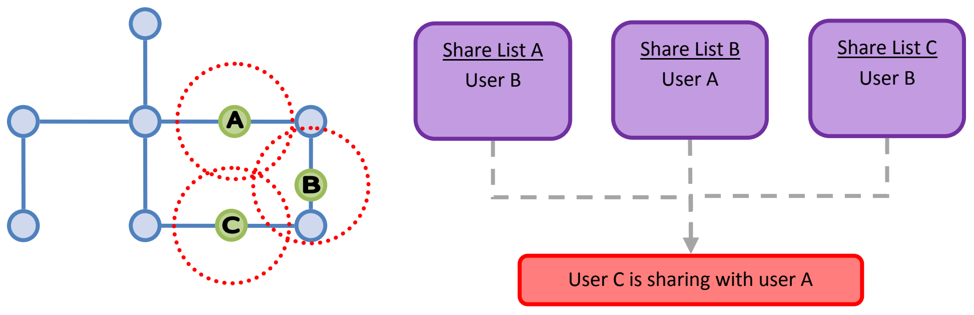


Figure 21: Sharing algorithm

5 Architecture

The architecture of the DMCA and the design of the DMCA and the used algorithms will be discussed in the following two chapters. The architecture chapter will primarily show several UML diagrams, the requirements and constraints of the DMCA.

5.1 Conceptual design

In the architecture chapter the basic buildup issues will be discussed. How the system is build up out of different parts, the design of these parts and the reason why these issues are design in this manner. The essential purpose of the DMCA is that it provides a simulation in which all the algorithms, movements and actions are executable and viewable. The DMCA also provides the tools for altering the simulation. The basic build up of the simulation is illustrated in Figure 22.

Within the DMCA a simulated world is realized and within this simulated world a multi agent system resides. The agents in this multi agent system communicate with each other and when this communication is established the algorithms are triggered and have a result in the form of a new constructed Personal map.

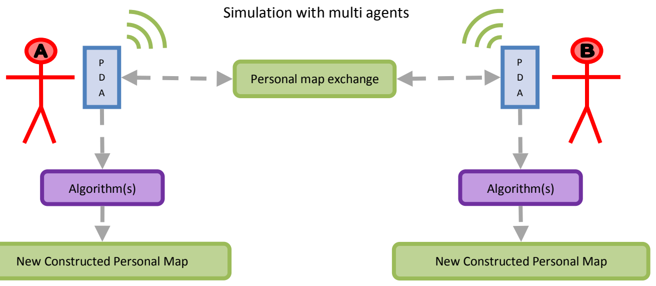

Simulation with multi agents
A
B
PDA
PDA
Personal map exchange
Algorithm(s)
Algorithm(s)
New Constructed Personal Map
New Constructed Personal Map

Figure 22: Conceptual design

The amount of agents in the illustration is set to two because the communication between agents is always one-on-one. This means that if one agent is communicating with another agent and a third agent wants to share its information, it will have to wait until the first two are finished communicating after which it can exchange its own information.

In actuality there will be many more agents in the simulation at one time so waiting for an agent to finish its communication will not occur but rather it will explore its own path further and will share its information with the first available agent it encounters.

Keeping this basic description about the conceptual design in mind the coming subparagraphs of the architecture chapter will reveal more on the functional designs of the DMCA and will show the corresponding unified models.

5.2 UML

The UML standard stands for **U**nified **M**odeling **L**anguage and is a system of diagrams in which the workflow of a system is represented from multiple points of view. These multiple points of view are represented by three different models of the same system:

- The functional model;
- The object model;
- The dynamic model.

Within the UML standard the functional model is represented by use case diagrams. These use case diagrams specifies the system functionalities from the users perspective. This entails that each action the user can perform within the system are modeled in this functional model.

The object model is represented by Class Diagrams. These class diagrams describe the structure of the system in terms of objects, attributes, associations and operations. The questions you should be asking yourself when building a class diagram for the object model are;

- Which objects does it contain?
- What are the objects attributes?
- Do the objects have any associations with each other?
- What operations can the objects perform?

The dynamic model is represented by sequence diagrams, state chart diagrams and activity diagrams. These describe the internal behavior of the system. The dynamic model shows the collaborations among the objects and changes the internal states of the objects.

Both the functional model and the object model will be represented in this report by respectively the use case diagrams of and the class diagram. The dynamic model however will only be shown partially, that is to say only the activity diagram will be displayed.

All the models are based on the DMCA, the algorithm(s) which are the main focus of this report, will be discussed in the next chapter.

5.3 Use case diagram

The use case diagram is used for the examination of the requirements for the application and the analysis of the application functionalities. The use case gives an external view on a specific segment of the system or the entire system at once. A use case element describes a function provided by the system that yields visible results for an actor. An actor represents any entity that interacts with the system (e.g. a user, another system, the systems physical environment).

5.3.1 DMCA use case

In Figure 23 the use case diagram of the DMCA is displayed. This diagram illustrates clearly that the DMCA main window contain four distinct segment s:

- Simulation;
- PDA;
- Options;
- Toolbar.

From the diagram we can see that the simulation only provides the view on the simulation and nothing more. The other segments however do provide multiple functions.

The PDA segment has three options which all augment the amount of active PDA in the simulated world. These functions include adding a new PDA to the simulated world, removing the selected PDA from the simulated world and removing all the PDAs from the simulated world at once.
The options for the simulation are divided into three segment s were the log segment only provides the view on the log but the remaining two (Rendering and simulation) add more information to the simulation, give information about the simulation and can influence the simulation when altering the values.

The simulation provided under options provides information about the simulation which is running in the DMCA. It provides the state of the simulation, the name of the xml files and the amount of PDA's currently residing in the simulation.
The rendering segment of the options can add or remove information from and to the simulation. Adding and removing the edge numbers, the node numbers, the PDA numbers and showing the PDA transmission range are the functions which add more information to the simulation. The adjustment of the PDA transmission range does not add more information to the simulation but influences the simulation.

The toolbar provides four options; two of these options influence the simulation in the way that we can pause and resume the simulation. The other two options can respectfully start the map creator and open a new xml map into the DMCA.

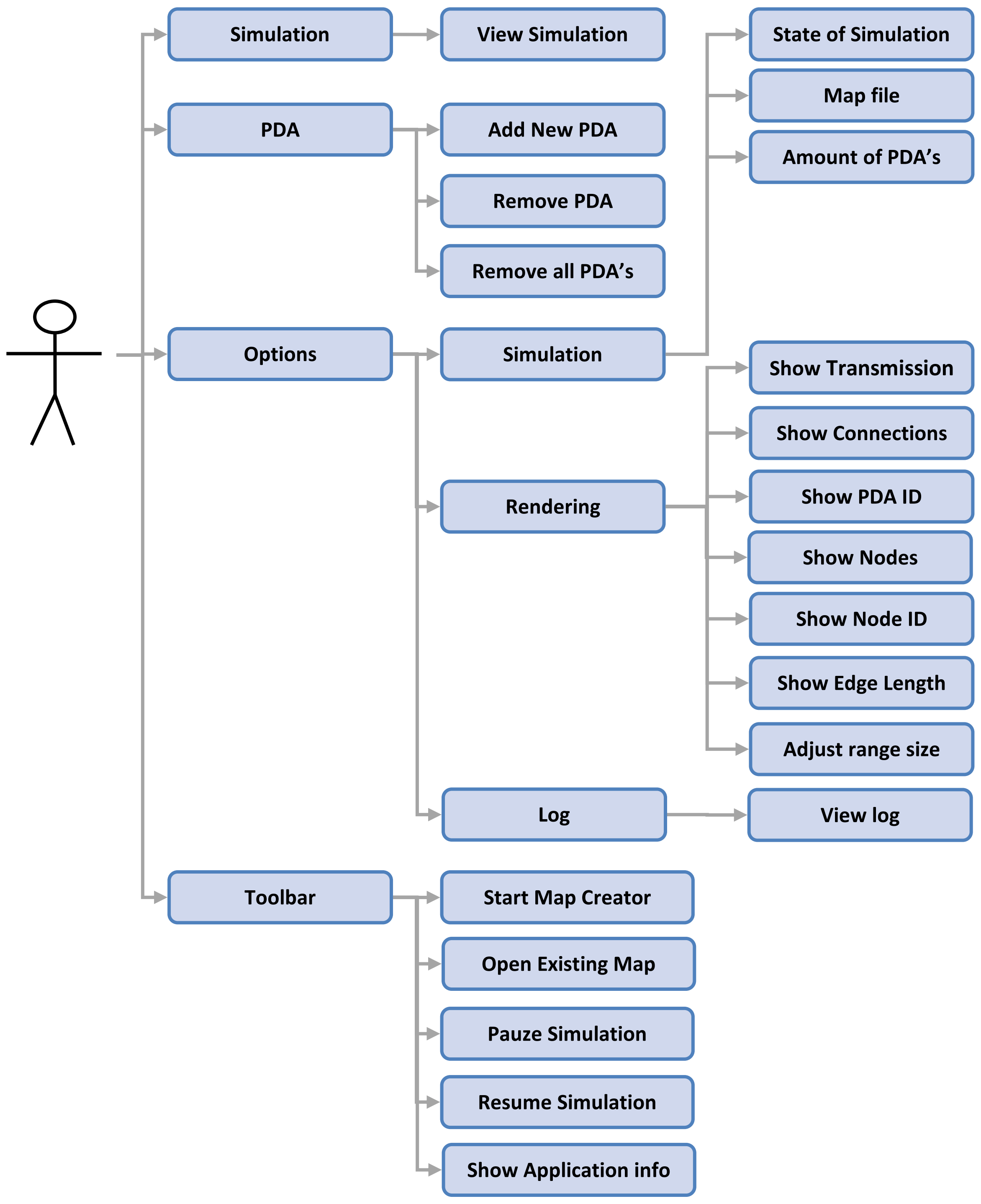


Figure 23: Use case diagram main window

5.3.2 Personal world view use case

In the next use case diagram illustrated in Figure 24, the Personal world view is displayed. The personal world view has two separate segments. The toolbar and the personal view window. The toolbar is a very basic toolbar from which all the functions can used. Naturally it has the close function with which the personal view window can be closed and the deactivation-reactivation button with which we can deactivate and reactivate the exploration algorithm.

It also has the function to show the explored personal map in a textual format, but this option is used for experimental purposes only and does not have any influence on the results. The last option available on the general toolbar is the zoom function. Two buttons are situated for zooming in and zooming out for the likely event that the map will become too large to display the map properly.

The world view only has one function and that is to displays the traveled path of the current agent. The zooming function naturally has an effect on the viewing window but the view itself only has the function to display the traveled path.

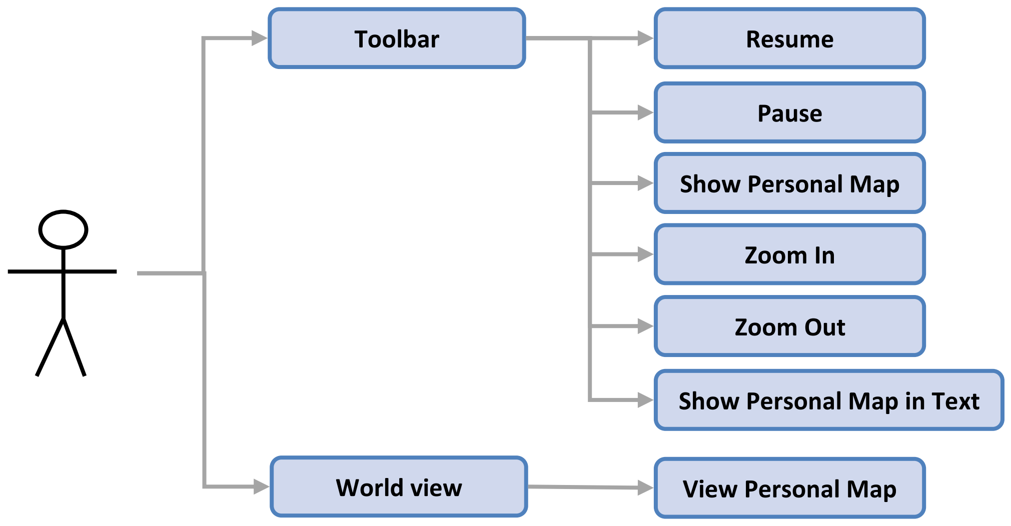


Figure 24: Use case diagram personal world view

5.3.3 Map creator use case

The last use case diagram for the map creator is illustrated in Figure 25. The map creator has a very basic design but has a lot of options. The toolbar for example provides a lot of different functions of which most of them can be executed and viewed on the canvas. The drawing of new line segments and the additions of blockades can for example be viewed directly after application.

Naturally not every function of the toolbar is intended for the canvas. The Save current map and close are functions which do not influence the canvas in any way but do contribute to the creation or augmentation of a map.

The canvas segment provides a real time view on the current map and all the augmentations the user makes when creating the new map.

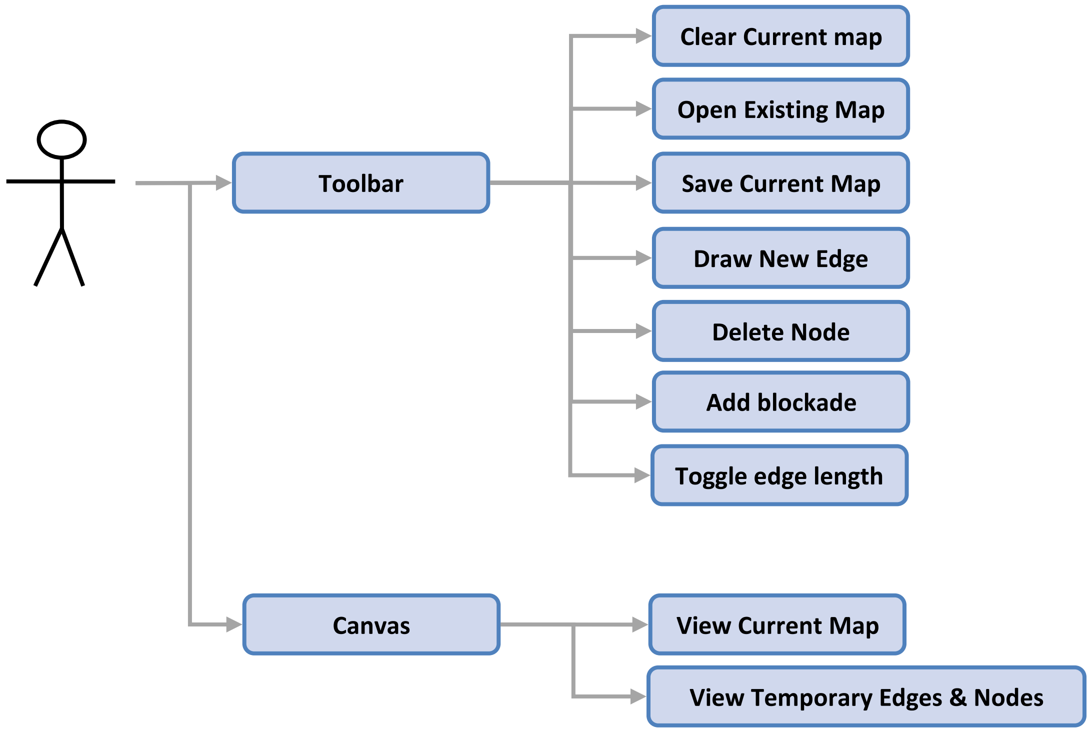


Figure 25: Use case diagram map creator

5.3.4 Class diagram

The class diagram shows the relationship between the different programming classes and structures in the DMCA. All the classes in this diagram have their own methods, properties and variables but not everything is displayed in this diagram due to space issues, however the main elements are displayed.

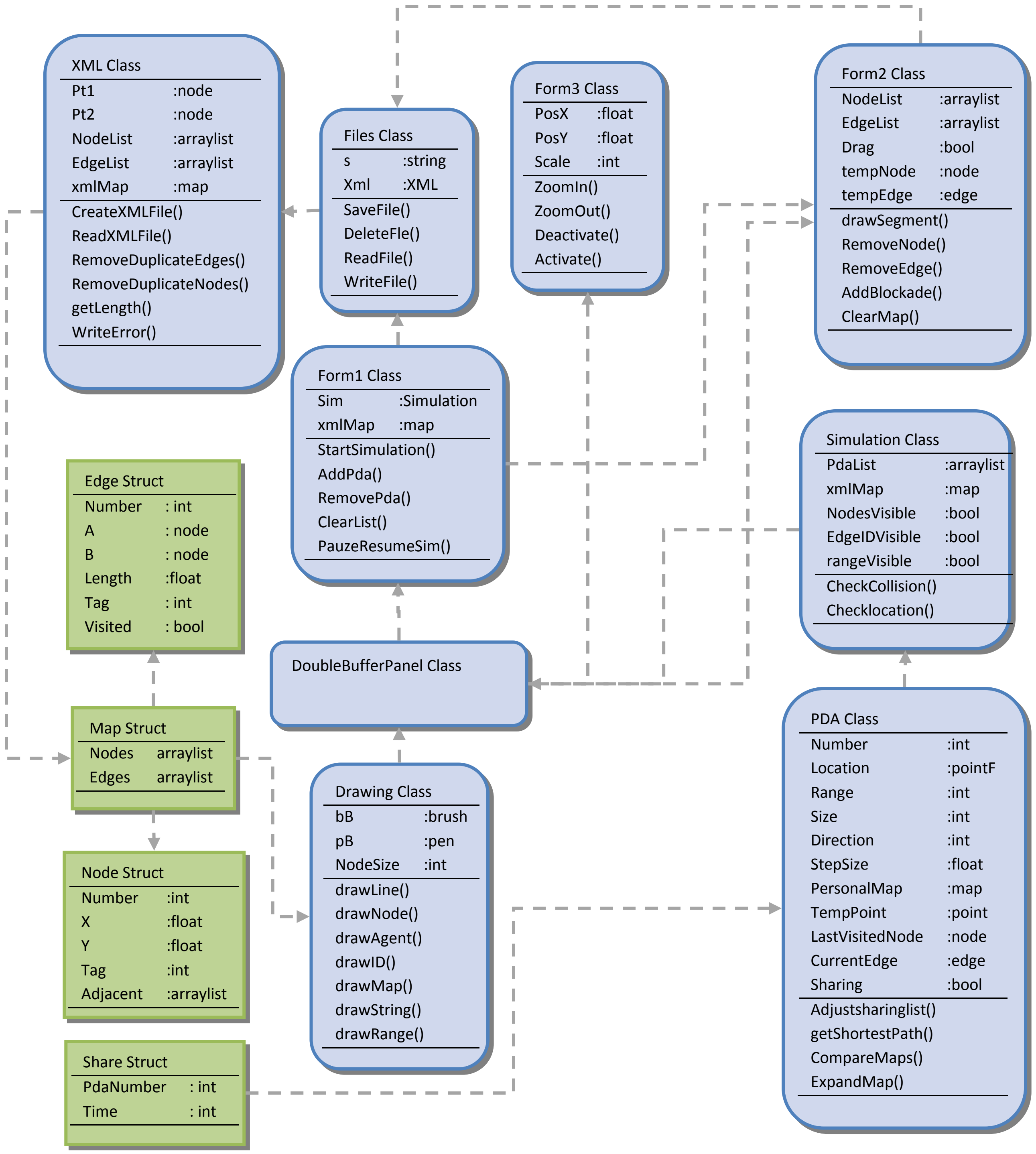


Figure 26: Class diagram DMCA

5.3.5 Activity diagram

An activity diagram is a diagram that shows activities and actions, this helps to describe workflows. In the Unified Modeling Language (UML) an activity diagram represents the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. To maintain the entire activity diagram of the DMCA in an orderly manner, the two red activities have their own activity diagrams which are illustrated by Figure 28 and Figure 29.

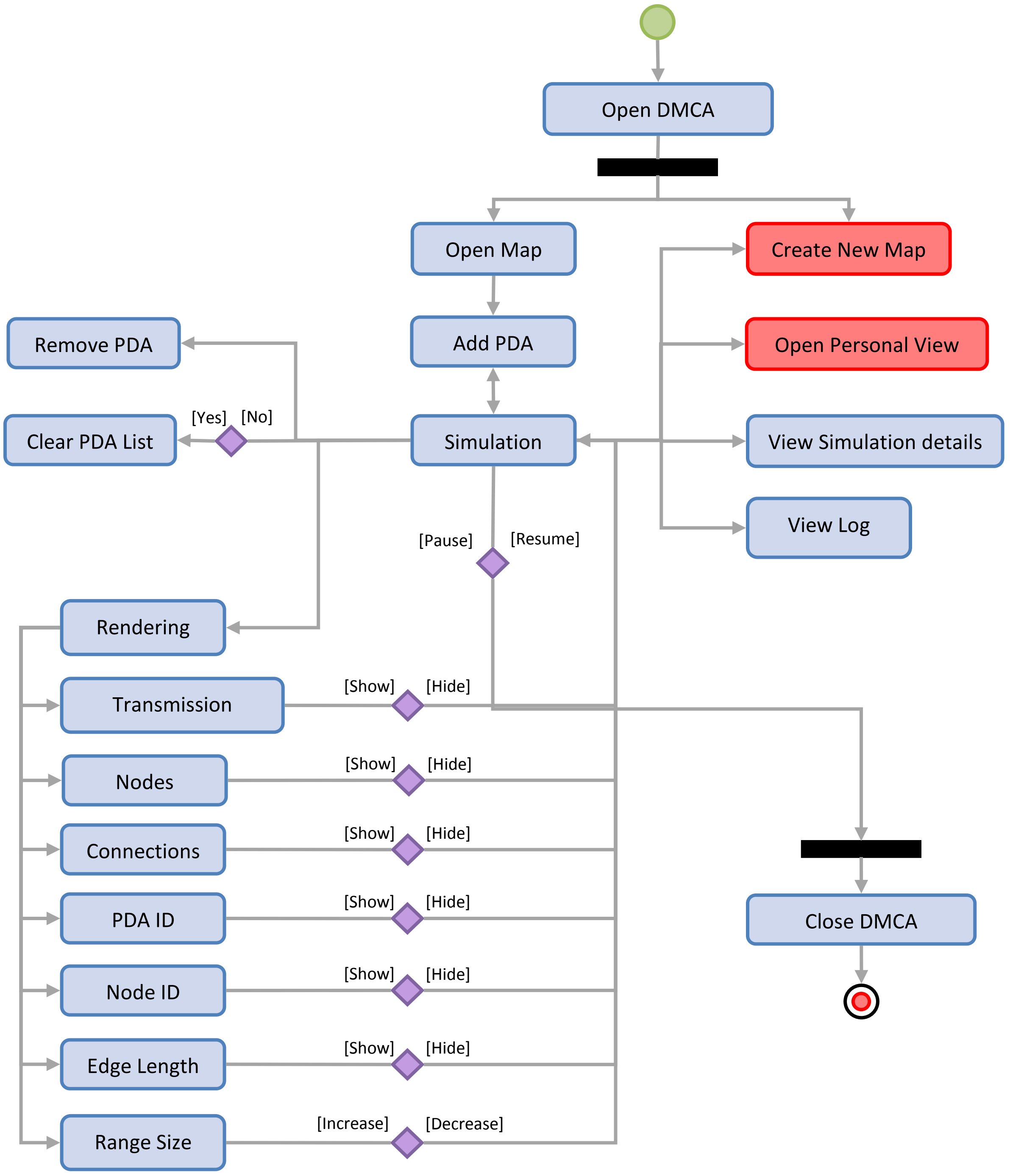


Figure 27: Activity diagram DMCA main window

Figure 28 illustrates the activity of creating a new map in the DMCA and all the options which are available in the process. The only thing which can be a little confusing in all the activity diagrams is that it is possible to end the main activity at every time in the workflow.

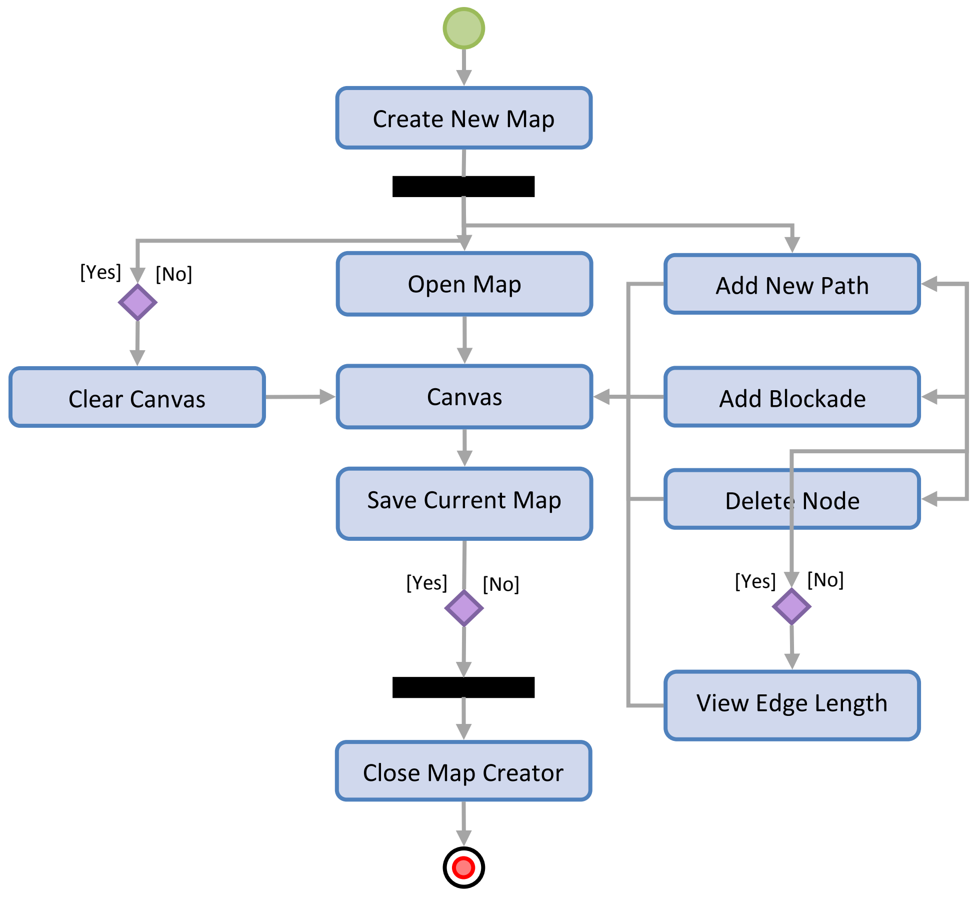


Figure 28: Activity diagram map creation

In this activity diagram it is clearly visible that the creation of a new map or the augmentation of an existing one has three main elements:

- The addition of a new path (two nodes with a connected edge);
- The addition of a node blockade;
- The deletion of a node (the node and the connected edges are removed).

All the other activities are used for opening, saving and clearing the map.

The last activity diagram shows the personal view activities. From Figure 29 it is very simple to conclude that the personal view has limited interactive abilities and has the main purpose to display the personal map. From this point it is possible to interact but only limited, pause the activities and zoom in and out to create a better view on the personal map.

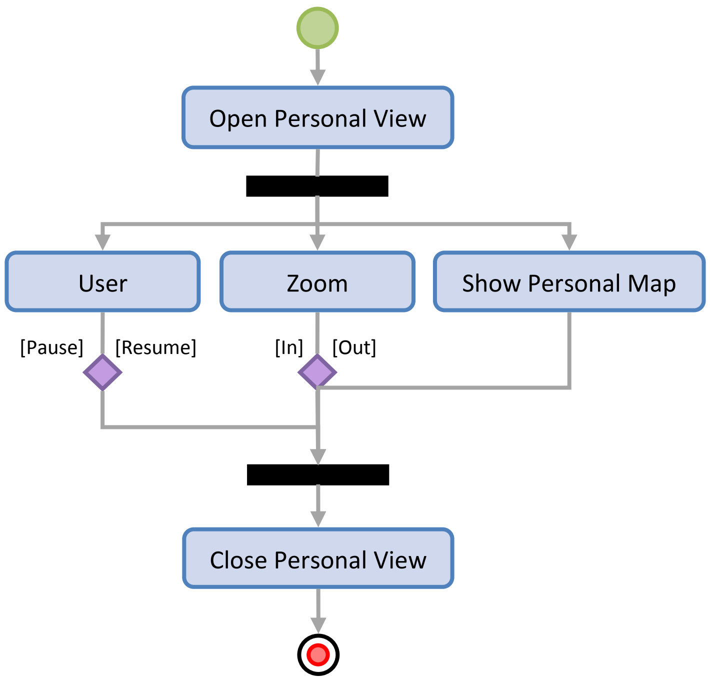


Figure 29: Activity diagram personal view

6 Implementation

All the elements which are needed to construct the DMCA and the construction of every object used in the DMCA will be displayed in the implementation chapter. All the tools will be briefly mentioned, and the program layout and some screenshots of the DMCA will be displayed as well.

6.1 Tools & Languages

The implementation of the DMCA was done with the programming language C# in the Microsoft Visual studio 2008 (with service pack 1) development environment. Both the C# language and the visual studio environment are developed by the Microsoft cooperation and the combination of both can create a very powerful implementation.

The choice to use the following techniques and tools are based on the personal experience of the developer with the develop environment, other positive features where the developers existing knowledge of XML and the relative flat learning curve of the C# language.

6.1.1 Microsoft Visual studio

The Microsoft Visual Studio development system is a suite of development tools designed to aid software developers whether they are novices or seasoned professionals face complex challenges and create innovative solutions. Visual Studio's role is to improve the process of development and make the work of achieving breakthroughs easier and more satisfying. [6]

At the beginning of development the 2005 version of the Microsoft visual studio was used but halfway into development the switch was made to the 2008 version. This did not bring any changes in the program code but did provide some more easy to use function which could speed up development.

6.1.2 C# language

C# (pronounced C sharp) is a multipurpose programming language which enables the user to build rich, connected web and client applications on the .NET Framework. The language is an object orientated programming language (OOP). Object-oriented programming is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.

The C# language is developed by Microsoft as part of the .Net framework and the programming language is based on C++ with elements from Visual Basic and Java. Like Java, C# provides automatic garbage collection, whereas traditional C and C++ do not.

[6] Text taken from the Microsoft website

6.1.3 Graphic Device Interface

The graphic aspect of this project is done with a graphic component of the C# language called the **G**raphic **D**evice **I**nterface (GDI+). The GDI+ component is responsible for the drawing of lines, curves, rendering fonts and handling pallets. GDI+ is the successor of GDI and has improved 2D graphics by adding advanced features such as:

- Anti-aliased 2D graphics;
- Floating point coordinates;
- Gradient shading;
- More complex path management;
- Intrinsic support for modern graphics-file formats like JPEG and PNG;
- General support for composition of transformations in the 2D viewing pipeline.

The GDI+ component is very easy to combine with the C# language en is powerful enough and has the right options to support the implementation with the needed elements.

6.1.4 Extensible Markup language

In addition to the use of the C# programming language and Microsoft visual studio, the e**X**tensible **M**arkup **L**anguage or more commonly known as XML was used as the format to store the map files.

XML’s design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML’s design focuses on documents, it is widely used for the representation of arbitrary data structures

The reason for choosing the XML standard instead of a simple text based file storage is based on the fact that an XML file is very capable to transport and store data. An XML file is build up of hierarchical modules which makes it very easy to construct an organized file and search the file for data. In the DMCA the XML files are used to store the map of the 2D world in, and to read from when loading a file into the DMCA.

6.2 Program layout

Naturally the layout design was not made instantly but is the result of a process of tryouts and tests. During the design of the implementation layout, it was clear that the application will have to have two separate parts. And that both parts must be available within the application at all times. This is why the implementation was split up into two parts:

- **Map creator:** The window for creating a new map;
- **The DMCA main simulation window:** The Main window for running a new simulation and augmenting the simulation in real time.

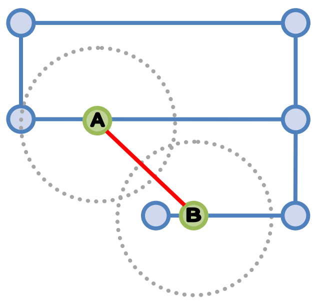


Figure 30: General situation

Both the Map creator and the main simulation window have some specific design details which were taken into account when designing the layout.

For the Map creator these were:

- All the new nodes have to be connected, so a new addition to the map consists out of two nodes with an edge in between;
- Nodes can be deleted and with it the connected edges;
- Editing existing map is an option (optional);
- Adding tags to points and edges;
- Saving the new map as a XML file.

For the DMCA main simulation window these were:

- The simulation must be visible at all times;
- The simulation environment view will have to be adjustable in real time;
- The simulation can be paused and resumed;
- PDA's can be added and delete in real time;
- PDA's world view must be viewable.

Naturally these two separate parts also have commonalities when it comes to design details. A general detail they both have is that all the available actions are directly accessible with the help of simple iconic buttons and that all the available actions are applied with the use of the common pointing device "the mouse".

The simple interface and a global understanding of what the program is suppose to do provides the application with a high learn-ability rate and can be operated with little or no experience.

6.3 Elements & objects

The XML files itself and the build-up will be the main element which will be reviewed. Together with all the introduced objects this element will be the main subject of this chapter.

6.3.1 XML files layout

The DMCA uses XML files to store its maps in. This XML file contains the basic building block for the entire simulation map and is used as input when loading the file into the DMCA. All the other elements like edge length, the edge visited value and the node's adjacent edge list are respectively calculated, updated real time and filled after the XML file is loaded.

The actual design of the XML files is shown in Figure 31 and the visual representation of the XML file is shown in Figure 32.

```
<? Xml version="1.0"?>
<Map>
<Nodes>
        <Node number="0">
                <x>75</x>
                <y>104</y>
                <NodeTag>0</NodeTag>
        </Node>
        <Node number="1">
                <x>353</x>
                <y>104</y>
                <NodeTag>0</NodeTag>
        </Node>
</Nodes>
        <Edges>
                <Edge number="0">
                        <From>0</From>
                        <To>1</To>
                        <EdgeTag>0</EdgeTag>
                </Edge>
        </Edges>
</Map>
```

Figure 31: XML program code

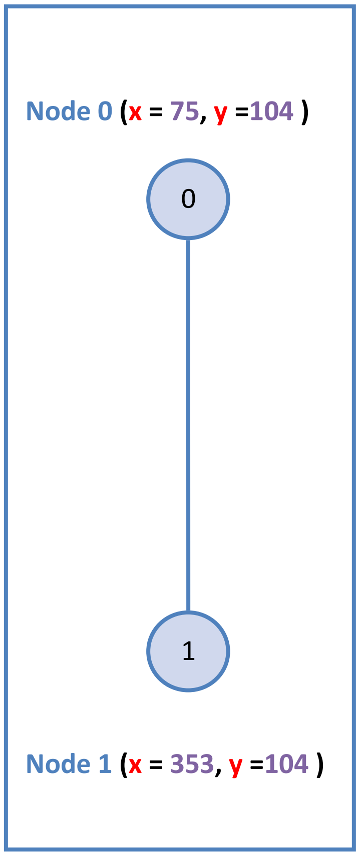


Figure 32: Visual representation

The design of the XML file is easily expandable by adding a new Node structure between the Nodes tags and adding a new Edge structure between the Edges tags. When expanding an existing map or making a new map in the map creator all the new points and new paths are added to a temporary object list and all the augmentations of the map are altered in this temporary list. After the user finalizes the map by saving it, the temporary object list is converted to a new XML file.

6.3.2 Node & Edge construction

Our graph data structure, just like any other graph data structure, exists of nodes and edges. The construction of these two elements is very common to other graph construction but has some custom elements in it. In Figure 34 and Figure 33 the construction of both the node and the edge are displayed. A short description of every part of the node and edge construction is displayed below in Table 2.

```
public struct Node
{
        int number;
        float X;
        float Y;
        int tag;
        ArrayList AdjP;
}
```

Figure 34: Node construction

```
public struct Edge
{
        int number;
        Node A;
        Node B;
        float length;
        int tag;
        bool Visited;
}
```

Figure 33: Edge construction

Table 2: Node construction description

The node construction exists of	
Node number	the numerical value of the node designation (Starts at 0).
Node X-value	The numerical value of the node's x-position in the 2D simulation world.
Node Y-value	The numerical value of the node's y-position in the 2D simulation world.
Node tag	A numerical value which determines the property this node has. A property of a node can vary from an exit, a staircase, a blocked node or another imaginable property.
Node adjacent edges list	A list of all the connected edges of this node.

The elements for the edge construction are also display in Table 3. Again all the elements have been accompanied with a short description.

Table 3: Edge construction desciption

The edge construction exists of	
Edge number	The numerical value of the edge designation (Starts at 0).
Node A	Node A is one of the nodes which connects too an edge. Because the edges are undirected the 'A' node can be the beginning or the end of the edge.
Node B	Node B is one of the nodes which connects too an edge. Because the edges are undirected the 'B' node can be the beginning or the end of the edge.
Edge Length	The length of the edge between node A and node B.
Edge Tag	A numerical value which determines the property this edge has. The property of an edge can only vary from blocked or not blocked in this case 0 or 1. The choice for a numerical value above a Boolean value is based upon the eventual extension of the tag possibilities.
Visited Value	This is a Boolean value which tells us if the current PDA has already visited this edge.

With the help of these short descriptions and Figure 35 a very obvious relation between the edge and the node can be constructed. The edge exists of two nodes with a connected path, the node contains a list of all its connected edges and because it has a list of its connected edges it also contains all the designations of its connected nodes.

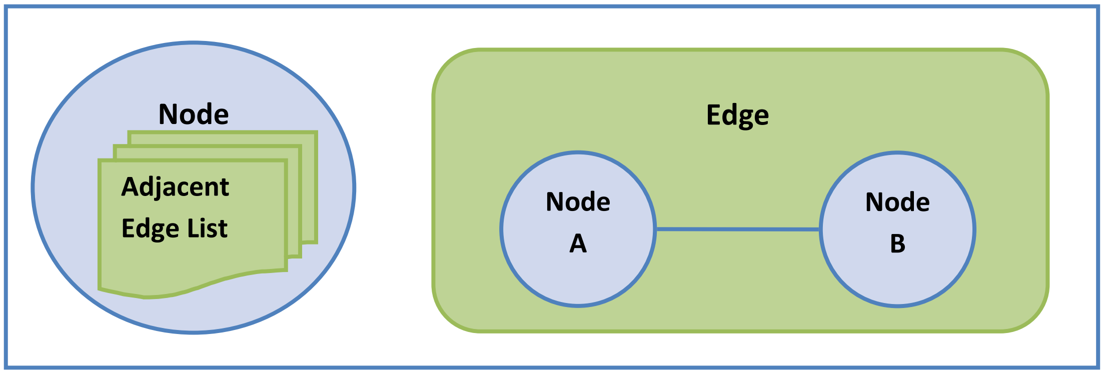


Figure 35: Node and edge relation

Because some restrictions and preconditions have been set at the beginning of this project, the amount of possible configuration of nodes and edges is limited. The restriction of the path angles and the vertical and horizontal directions reduces the amount of different nodes to only 15 possible configurations. These are divided into four categories, and all four of these categories are displayed in Figure 36.

Every category displays its possible configuration of node and connected edge(s). The two edges category is a special case because it has two different shapes for its possibilities, that is why a distinction is made between the two shapes by giving them a type 1 and type 2 designation.

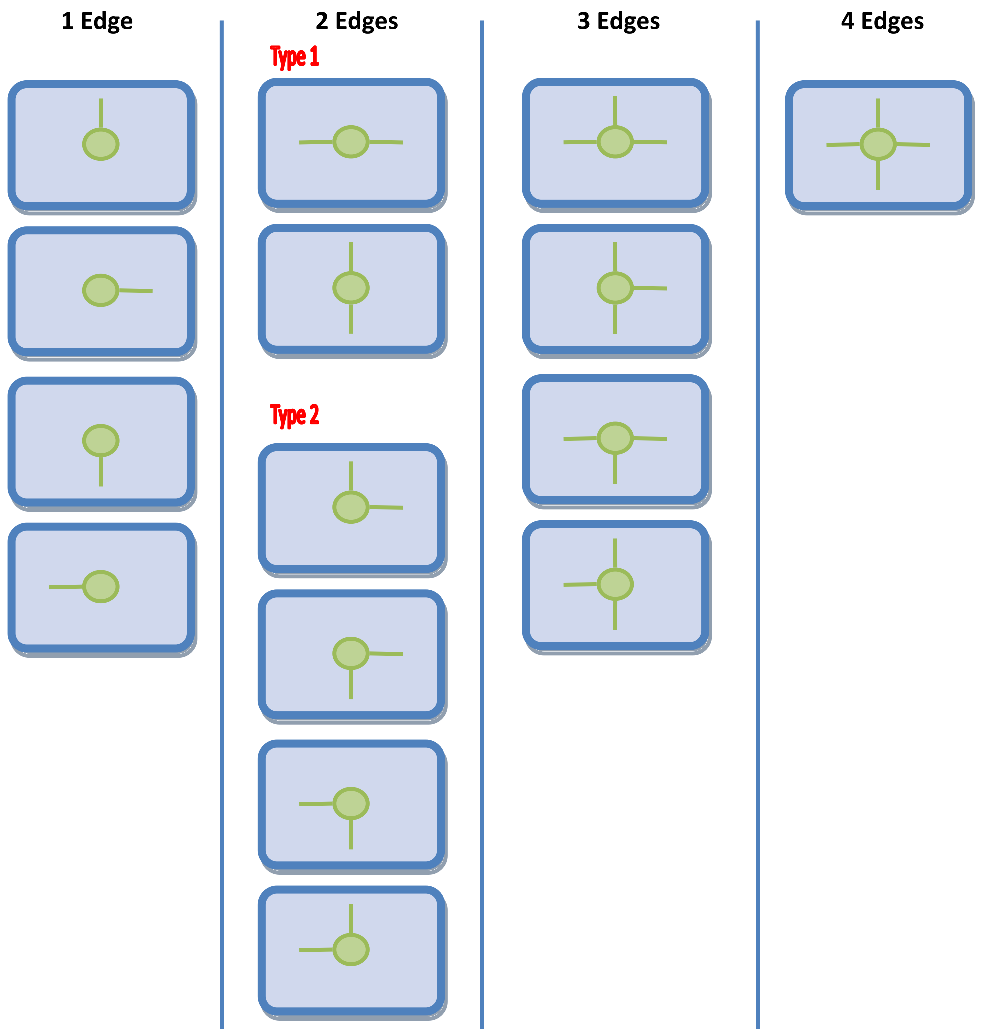


Figure 36: All types of nodes

The node with four connected edges is only displayed once because it has only one possible construction. However it has four different positions because it can be rotated 90^o 180^o or -90^o, the construction of the node will not vary but it definitely has four different possibilities.

6.3.3 Map construction

The map construction is quite simple; it just contains two different lists one list with all the Nodes and one list with all the edges. The edge list is needed because only keeping a list of all the nodes will results in a large quantity of duplicate edges. To visualize the entire map both these lists will be loaded and displayed in the viewing canvas.

The map data within this file consists of:

```
public struct Map
{
    ArrayList Nodes;
    ArrayList Edges;
}
```

Figure 37: Map contruction

6.3.4 PDA construction

The PDA construction will function as the agent in the virtual world. Agent or user would probably be a more appropriate designation but because the personal map and the calculations for the new personal map are being showed and executed on a virtual PDA, this designation was chosen.
To elaborate more on the PDA construction, a short description will again be given of all the element of the construction which is shown in Figure 38.

```
public class PDA
{
    int number;
    PointF location;
    int range;
    int size;
    int direction;
    float StepSize;

    Map PersonalMap;
    Point tempPoint;
    Node LastVisitedNode;
    Edge CurrentEdge;
    bool Sharing;
}
```

Figure 38: PDA construction

Table 4 shows all the properties of the PDA and provides a short description on these properties. Some of these properties are dynamic, others are very static or are only set once during the whole simulation, usually at the beginning.

Table 4: PDA properties

The PDA properties	
PDA number	The numerical value of the PDA designation (Starts at 0).
PDA location	The floating point object which contains two numerical values for the X- and Y-value. The X-value and the Y-value represent the x- and y-position in the 2D simulation world.
Range	The numerical value for the size of the communication range of the PDA.
PDA Size	The numerical value for the size of the PDA in the simulation. This is a static value set to the size of 10.
PDA direction	Direction is the numerical value for the direction in which the PDA is going. The direction varies from 0 to 3 with 0 = East, 1 = West, 2 = South and 3 = North.
PDA stepsize	The PDA StepSize is the numerical value for the traveling speed of the PDA trough the graph. This is a floating point value and has 4 different values which all end up in an integer value when we are adding them. These values are: 0.1, 0.2, 0.5 and 1.0.
Personal Map	The personal map is a Map object (as see in paragraph 5.2.3). The Personal map contains the personal data which the PDA collects when traveling in the undiscovered world.
Temppoint	The tempPoint is a Point object with a X-value and a Y-value in it, these represent the X- and Y-position in the 2D simulation world. The tempPoint is the current position of the PDA in the simulation.
LastVisitedNode	The LastVisitedNode is a node object and is the last node the PDA has visited.
CurrentEdge	The CurrentEdge is an Edge object and is the same object as the current edge on which the PDA is positioned.
Sharing	Sharing is a boolean value which tells if a PDA is sharing yes or no. When the PDA is not sharing it may receive new information from other PDAs.

6.4 Graphical user interface (GUI)

The **G**raphical **U**ser **I**nterface or GUI is the graphic representation of the application interface. The GUI purpose is to provide a functional and practical way of controlling the DMCA. The design of the GUI finds its origin in the GUI from the ManetLoc project [1] [23] but has a lot of different aspects and functions. The GUI has a central starting form, from which all the basic functions and options of the simulation can be reached. Most of the interaction with this form is done with the help of iconic buttons which clearly represents the function.

6.4.1 Distributed Map Construction Application (DMCA)

The DMCA will provide the simulated world and the exploring entities within this world. The simulated world will be visualized as a two dimensional graph in which the entities must reside. All the entities will keep their own traveled path and will share their personal map information to complete the entire world.

The personal maps of the multiple entities will be viewable, the match and merge algorithms for sharing personal maps will be used in the simulation and the results of these algorithm will also be viewable.

The entities in the graph will be implemented as a basic multi agent system. All the agents will have their own data and variables but will have similar reasoning and abilities. The agents in the simulated environment are used for the exploration of the undiscovered world. They will store personal data and exchanged them among each other to acquire more information.

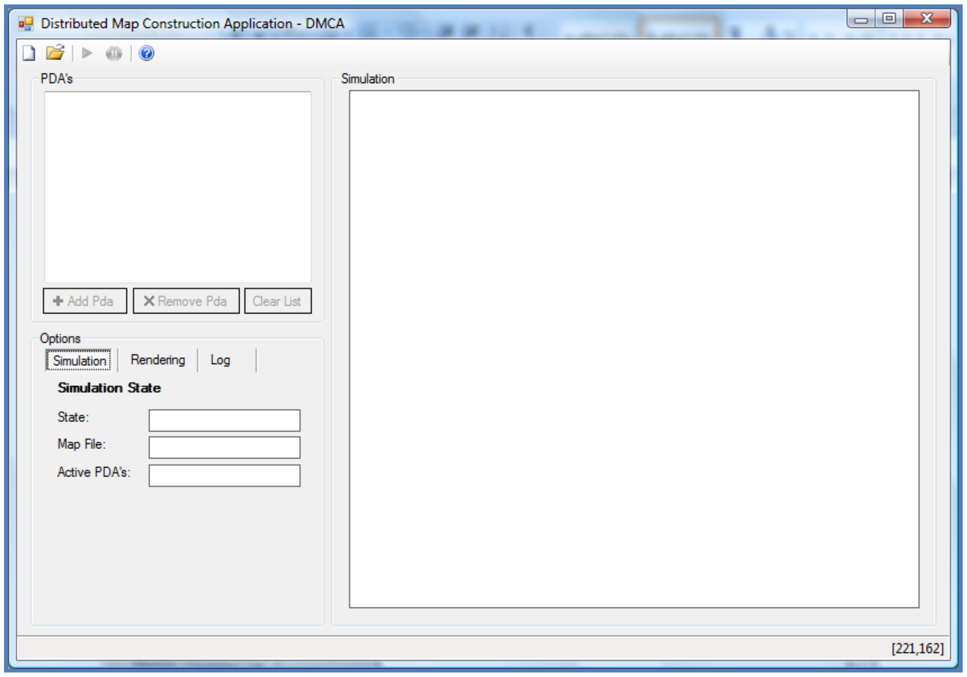
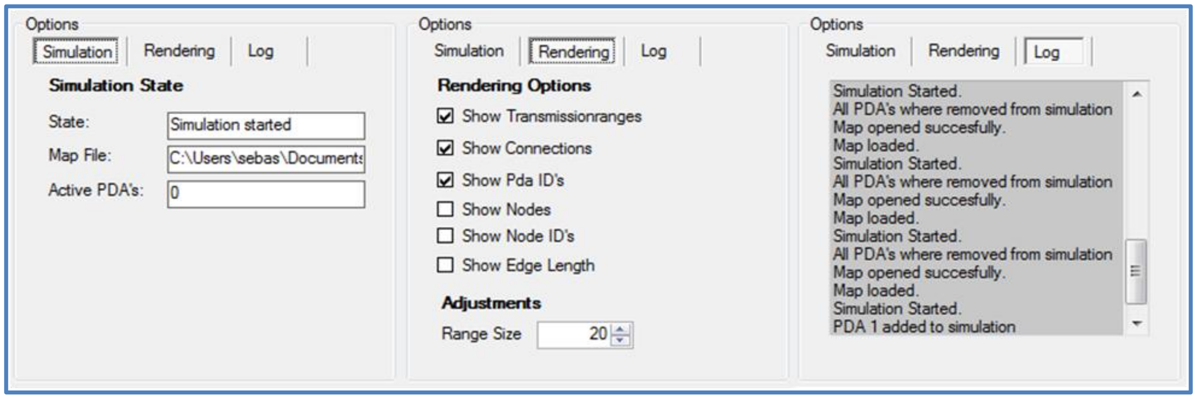


Figure 39: Main simulation window DMCA

In Figure 39 the main simulation window is displayed, this is the starting point of the DMCA and the point from which every aspect of the application is controlled. The map creator can be started from this point and the simulation can be viewed and edited in real time.

The options segment (displayed in Figure 40) controls different elements of the simulation but also displays the state the simulation is in, and the actions that have been taken. The rendering part of the options segment is the most useful of the three because it provides the most information and with the altering of the range size can influence the simulation in a significant manner.



Simulation State

State: Simulation started
Map File: C:\Users\sebas\Documents
Active PDA's: 0

Rendering Options

☑ Show Transmissionranges
☑ Show Connections
☑ Show Pda ID's
☐ Show Nodes
☐ Show Node ID's
☐ Show Edge Length

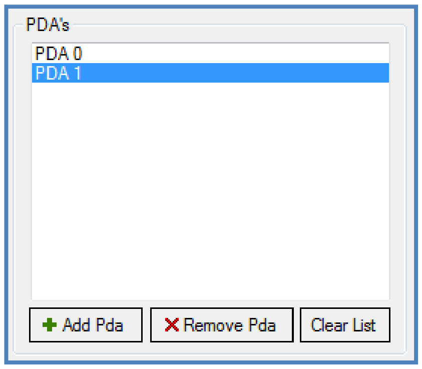
Adjustments

Range Size 20

```
Simulation Started.
All PDA's where removed from simulation
Map opened succesfully.
Map loaded.
Simulation Started.
All PDA's where removed from simulation
Map opened succesfully.
Map loaded.
Simulation Started.
All PDA's where removed from simulation
Map opened succesfully.
Map loaded.
Simulation Started.
PDA 1 added to simulation
```

Figure 40: Options segment

The PDA segment, in Figure 41 displays the current pda's which are in the simulation, it also provides the option to remove specific pda's or add a new one too the simulation. The third option here is to clear all the pda's from the simulation at once. Naturally this will remove all the data from the simulation and the simulation will have to start all over to collect new data. Double clicking the pda designation will open its personal view on the world, this option will be elaborated on in the next paragraph.



PDA 0
PDA 1

Figure 41: PDA segment

The toolbar on top of the DMCA (Figure 42) is the most basic part of the GUI but contains some essential elements. The map creator which will be disused in a following paragraph, for example is started with the blank map icon on the toolbar.

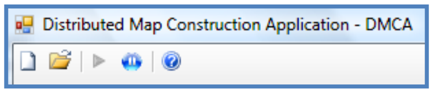


Figure 42: DMCA toolbar

A new map can also be opened from this toolbar, and the current simulation can be paused and resumed.

The last segment of the DMCA main simulation window is, the simulation view. This basically displays the entire simulated world and the agents/pda's which reside in the simulated world.

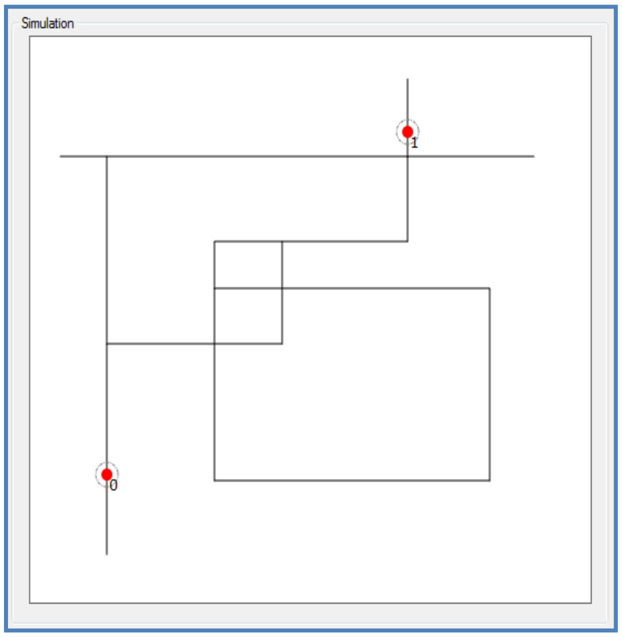


Figure 44: Basic simulation view

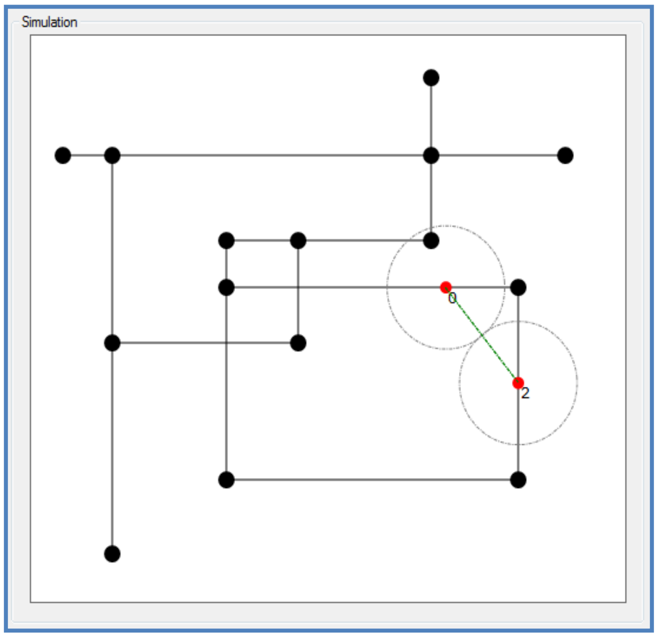


Figure 43: Altered simulation view

In combination with the rendering part of the options segment it can display alterations in the simulated world such as:

- Showing/removing the transmission ranges in the viewable simulation;
- Showing/removing the Pda connections in the viewable simulation;
- Showing/removing the Pda ID's in the viewable simulation;
- Showing/removing the node ID's in the viewable simulation;
- Showing/removing the nodes in the viewable simulation;
- Showing/removing the edge length in the viewable simulation;
- Enlarging the communication range.

Some of these alterations are displayed in Figure 43.

6.4.2 Personal world view

The personal view window is opened when the pda's designation is double clicked in the PDA segment of the main simulation window. The personal view window provides a graphic representation of the agent/pda's personal view on the explored and shared world. The personal view also provides a zoom function for the likely occurrence that a world is too large to display in the standard view port.

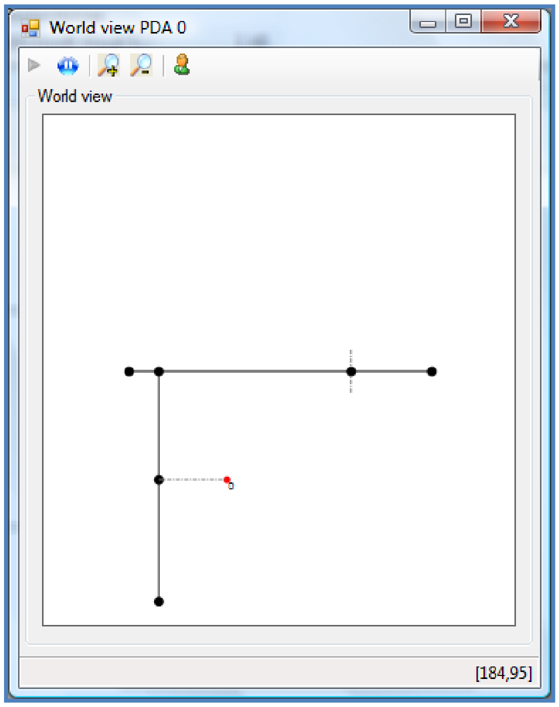


Figure 46: Personal view

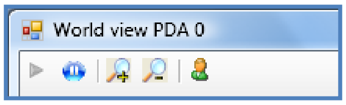


Figure 45: Toolbar World view

Naturally this function can zoom in and zoom out, so if a specific element of the personal view has to be examined, this function makes it possible.

The world view also provides the option to disable or enable the agent/pda's movement. This only disables or enables the movement and does not remove the agent/pda from the simulation. The agent/pda will still shares his information with the other agent's/pda's and receive information from them.

In the personal view the personal data can also be viewed. This will only display the nodes and edges which the agent/pda has visited. This option is only used for testing the application.

6.4.3 Map Creator

The map creator tool is used to create new maps to use in the main program as a simulated world. With this tool the user can not only create new world for the DMCA, but it also provides the option to edit existing maps. The map creator is displayed in Figure 47 and contains a variety of options.

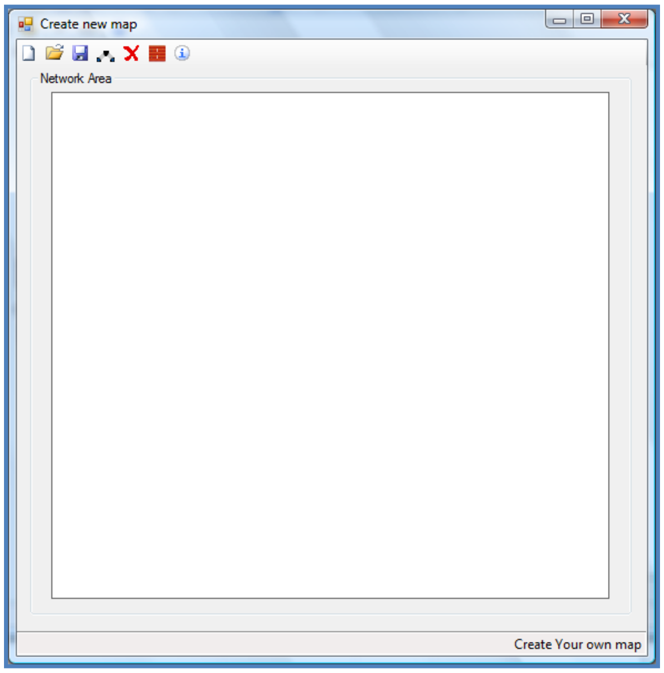


Figure 47: Map creator

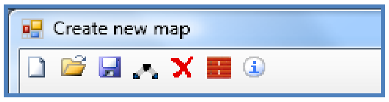


Figure 48: Toolbar map creator

The tool contains some basic options such as:

- Clearing the canvas
- Opening an existing map
- Saving the current creation
- Adding a new line segment
- Removing nodes
- Adding a blockade to a node
- Showing edge length

7 System Tests

Even though the system is a simulation it still needs testing to evaluate its performance and see if it achieves the pre-set goals. Other important aspects of the system which will be tested will be the GUI usability and completeness. And finally no software is without bugs and that is why only major bugs will be corrected and all minor defects will be listed as future work.

7.1 Test variables

Before starting the system tests first the variables with which the tests were conducted will have to be clear. In this paragraph all the variables and the reason why these variables were used will be elaborated on.

7.1.1 Maps

For the system test a variety of different maps have been created and used, to be more specific 3 different maps were used all with different amount of nodes. A map with 10 nodes, a map with 30 nodes and a map with 100 nodes were created and all have different shapes and do not have any relationship with each other.

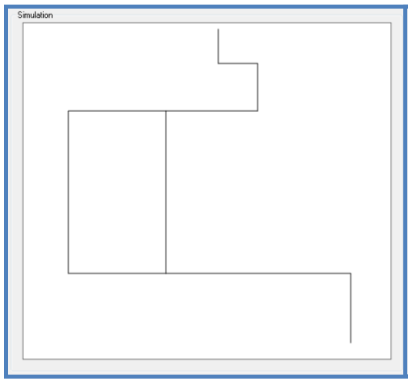


Figure 49: Ten nodes

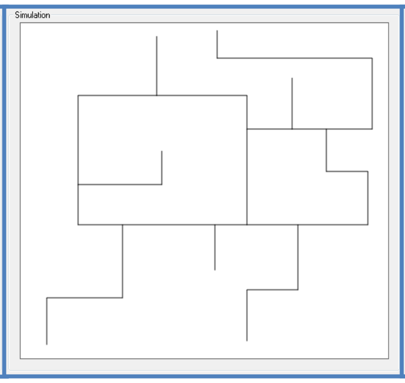


Figure 50: Thirty nodes

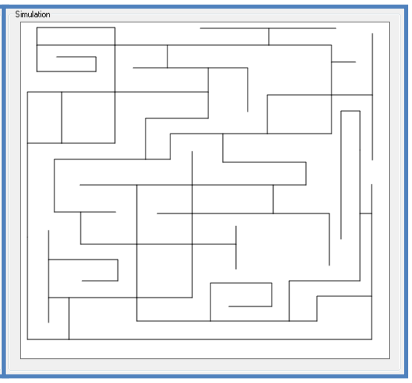


Figure 51: One hundred nodes

The fact that a map with a high and a low amount of nodes was used allows for tests with larger and smaller worlds and see if this influences the performance. Due to the amount of time it takes to test the world with 100 nodes this world was used less than the other two.

Needless to say is that during the implementation of the DMCA many other maps with a wide variety of configurations were used to test a variety of elements.

7.1.2 Exploration

Exploration is simply the manner in which the agents explore the unknown world and this aspect plays an important part in this project. When we look at the used algorithm(s) to match and merge the maps together we could adapt the exploration algorithm to maximize the match and merge algorithm.

The match and merge algorithm can only be executed when an agent has a minimum of four complete nodes in its personal map. This entails that the exploration algorithm should try to explore nodes one at a time. This means exploring an edge than traveling back to the start node and exploring a different edge.

However, this is not a very realistic situation when exploring an environment so the exploration algorithm is build up to handle 4 different situations displayed in Table 5.

Table 5: Exploration situations

Nr.	Situation	Action
1.	Agent encounters a node with multiple unvisited edges.	Travel on random unvisited edge.
2.	Agent encounters a node with one unvisited edge.	Travel on the unvisited edge.
3.	Agent encounters a node with no unvisited edges.	Look for unvisited edges in connected nodes, if found travel to connected node, if not found go to step 4.
4.	Agent encounters a node with no unvisited edges and does not have unvisited edges in its connected nodes.	Look for nearest node with unvisited edge and travel on first edge in travel path.

7.1.3 Hardware

Because this is a simulated test some of the results of these tests are dependent of the hardware upon which it is tested. Properties like the amount of time which an agent needs to explore the entire map are influenced by the hardware. For this reason the used hardware is displayed in Table 6.

Table 6: Test hardware

All the tests were performed on the following hardware	
CPU	Intel Pentium Core 2 Duo E6600.
RAM	4,00 GB 1333 MHz RAM.
Graphics card	NVidia GeForce 9600 GT.
Hard disk	Western Digital WD 740GD.
Operating system	Microsoft Windows 7 64 bits.

7.2 Correctness

The correctness test will determine if the DMCA processes its data correctly and whether or not it results in a correct conclusion, or in other words if the agents in the simulation create a correct personal map.

During all the tests with the 10, 30 and 100 node maps and with all the other maps in the final stages of the DMCA the correct and complete map was found without the use of any kind of shortcut or cheat. However I can't imagine that there isn't some kind of graph that would be completed incorrectly, but I just didn't encounter it.

7.3 Completeness

Completeness defines how the realized system compares to the related issues stated in the original problem definition in paragraph 1.2.

Related issues:

☑

- **The task of exploring the unknown world will be distributed amongst multiple entities.**
  Answer: The exploration task is distributed among all the agents in the simulated environment, We can consider this a multi-agent system and consider this related issue as completed.

☑

- **The individual entities will have similar behavior and can use and detect location information.**
  Answer: all the agents have the same level of intelligence and behavior.

☑

- **The environment has no pre-setup infrastructure and the entities in the environment do not have any knowledge of the environment.**
  Answer: A infrastructure less environment is easily achievable because a simulation is used, however in a real life situation it is not that difficult to achieve the same situation. Each agent does not have any preset knowledge of the environment and does not make use of any shortcuts or cheats to obtain more knowledge. It can only acquire knowledge from its own observations and trough communication with other agents. We consider this related issue as completed.

☑

- **This approach will be simulated and with the help of this simulation the algorithm(s) can be examined and tested.**
  Answer: The DMCA provides the simulation and the test results can be viewed in the following paragraphs so we consider this related issue as completed.

☑

- **The unknown world will be represented by a graph.**
  Answer: This was the simplest issue to complete because it is the base structure from which the maps and the environment are build up from.

☑

- **The test results of the simulation will be examined and will provide us with information on the algorithm(s), in what way we can improve the algorithm(s) or in what way has the algorithm(s) failed its purpose.**
  Answer: The test results can be viewed in the following paragraph

7.4 Performance

When testing with multiple agents, the agents will have the same traveling speeds and have the same transmission range. Although the equal speed will create a risk of cluttering, it is needed to do a time test. To test these maps a number of simulation has been executed to see is the algorithm work correctly, efficiently and where the eventual problems occur. Twenty five simulation will be executed for each map, these twenty five simulations are divided in five simulation with a certain amount of agents in the simulation. First five with one agent and the second five with two agents and so on. The first simulation with only one agent will act as a benchmark for the highest amount of time it take one agent to explorer the entire map. A comparison will be made with the simulations with multiple agents in them and the time it takes the first agent to construct a complete view of the entire map.

7.4.1 The 10 node map

All the results from the simulation with the use of the map with ten nodes.

Table 7: data from the 10 node map

Map	Agent(s)	Run(s)	1e Complete	All Complete
10 nodes	1	1e	1:54.9	1:54.9
10 nodes	1	2e	1:49.8	1:49.8
10 nodes	1	3e	1:46.2	1:46.2
10 nodes	1	4e	1:58.0	1:58.0
10 nodes	1	5e	2:04.4	2:04.4
10 nodes	2	1e	2:14.0	2:15.2
10 nodes	2	2e	1:55.8	2:00.0
10 nodes	2	3e	1:45.1	2:22.2
10 nodes	2	4e	1:42.7	1:52.0
10 nodes	2	5e	1:42.8	1:47.4
10 nodes	3	1e	1:39.5	1:50.1
10 nodes	3	2e	1:44.9	2:04.8
10 nodes	3	3e	1:28.0	1:42.1
10 nodes	3	4e	1:38.4	1:56.3
10 nodes	3	5e	1:39.5	1:47.3
10 nodes	4	1e	1:38.8	2:04.6
10 nodes	4	2e	1:33.4	2:04.7
10 nodes	4	3e	1:32.9	1:43.1
10 nodes	4	4e	1:33.4	1:43.5
10 nodes	4	5e	1:41.8	1:59.4
10 nodes	5	1e	1:40.1	1:49.5
10 nodes	5	2e	1:43.3	1:53.4
10 nodes	5	3e	1:49.5	1:58.6
10 nodes	5	4e	1:21.7	2:01.3
10 nodes	5	5e	1:35.2	2:01.2

All these results are displayed 10 node map resultsFigure 52. Here we can see that the 10 node map does contain some improvements over the single agent exploration but that these are very minimal. The main fact for this is that the design algorithm uses an acceptance level of four complete nodes to determine if a correct match can be made [7].

However four out of ten nodes is a to high percentage to make a significant improvement in the map exploration. As the graph displays the dark blue line is the benchmark and the light blue line is the simulation with 5 agents. The 5 agent simulation should have the best results but even do, it too reaches a point where it's less efficient as the single agent exploration. The exploration and construction with the help of four agents is the most consistent and has an overall good result.

The number of simulation can play a factor here, naturally five simulation for each situation is not much but it does give a good impression of the direction in which the results will go. During development the simulations where tested very frequently so it should provide us with consistent results.

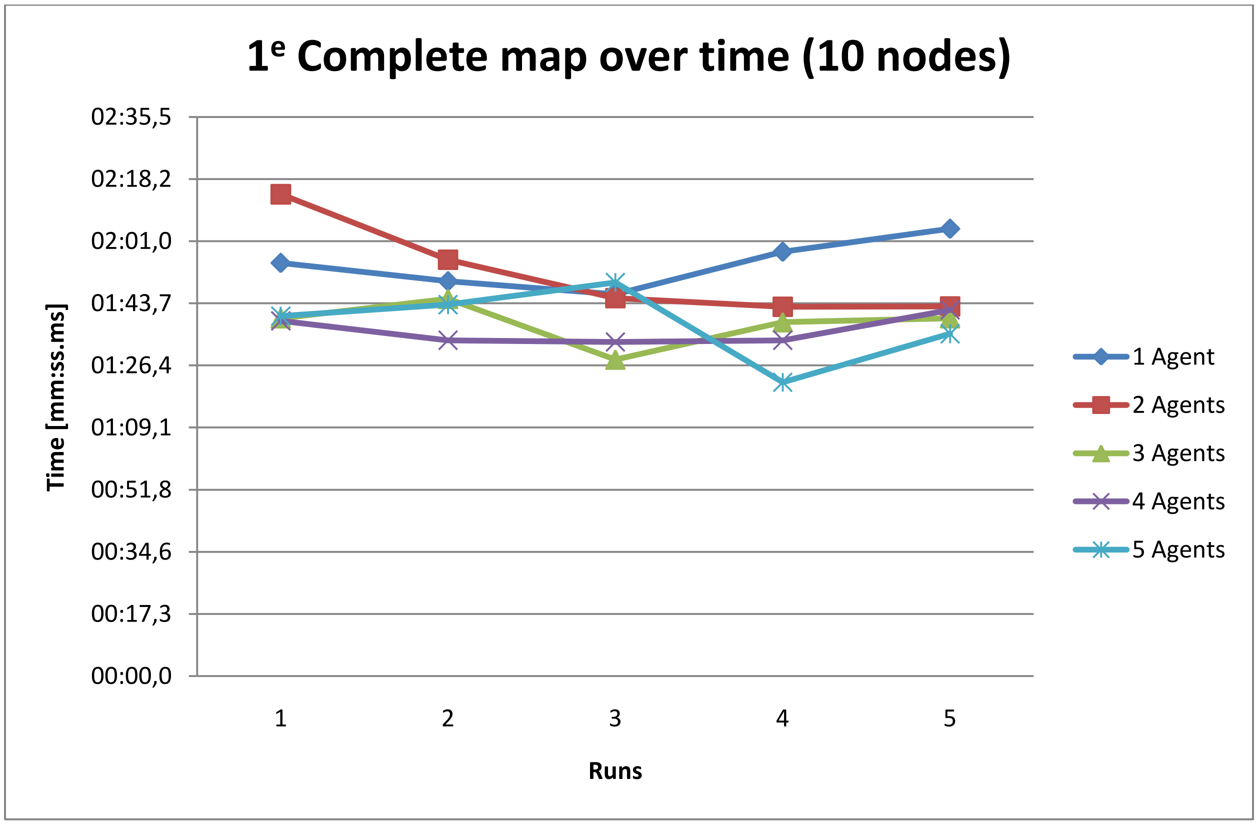


Figure 52: 10 node map results

[7] The algorithm is elaborated on in chapter 4.2

7.4.2 The 30 node map

The 30 node map is a better testing ground for the algorithm because the dynamics and the cooperation between the agents have more advantages. In this case the agent only has to have 13,33% of the entire map to make a valid match, where as in the 10 node map it would have to have 40%.

The results in the "All complete" column is a little misleading because the situation occurred a lot that all the agents would have a complete map with the exception of one. This last one was usually situated in a different part of the map and couldn't connect with the others for a while, and this would influence the results. These situations occurred more in the 100 node map but also occurred in the 30 node map.

Table 8: Data from the 30 node map

Map	Agent(s)	Runs	1e Complete	All Complete
30 nodes	1	1e	4:52.0	4:52.0
30 nodes	1	2e	4:29.4	4:29.4
30 nodes	1	3e	4:21.7	4:21.7
30 nodes	1	4e	4:39.3	4:39.3
30 nodes	1	5e	4:32.0	4:32.0
30 nodes	2	1e	3:10.0	3:14.7
30 nodes	2	2e	2:40.8	2:42.6
30 nodes	2	3e	2:41.3	2:50.8
30 nodes	2	4e	3:32.7	3:34.0
30 nodes	2	5e	3:46.4	3:52.3
30 nodes	3	1e	2:58.9	4:21.1
30 nodes	3	2e	2:27.9	2:30.2
30 nodes	3	3e	3:16.4	3:43.8
30 nodes	3	4e	2:37.8	2:40.5
30 nodes	3	5e	3:35.5	3:45.9
30 nodes	4	1e	2:29.1	3:17.5
30 nodes	4	2e	2:16.7	2:18.5
30 nodes	4	3e	2:44.4	2:49.7
30 nodes	4	4e	2:57.1	3:23.0
30 nodes	4	5e	2:57.2	3:09.7
30 nodes	5	1e	2:42.9	2:51.6
30 nodes	5	2e	2:30.2	3:10.8
30 nodes	5	3e	2:17.9	2:48.9
30 nodes	5	4e	2:07.3	2:20.6
30 nodes	5	5e	2:28.7	2:46.7

The results displayed in Figure 53 shows a significant difference between the single agent exploration and the exploration with multiple agents.

The simulations with two and three agents both have very varying results. The reason for this is that the combination of larger maps and a minimal amount of agents can create a situation where the agents do not meet each other or just meet each other once. This situation is actually very similar as a single agent exploration but with multiple agents, and will not have the same effect on the results that it should have had.

In the simulations with four and five agents the results are much more consistent and usually have the best results in comparison with the other simulations. With this amount of agents in the map and this size of the map, communication between the agents is guaranteed and this is very beneficial for the algorithm.

If the best results from the single agent exploration (4:21.7) and the five agent exploration (2:07.3) are compared, it shows a difference of two minutes fourteen seconds and four milliseconds (2:14.4). The time it took for the five agent exploration is less than half of the time it took the single agent to explore the entire map. This is efficiency increase of 100% and can be considered as a good result.

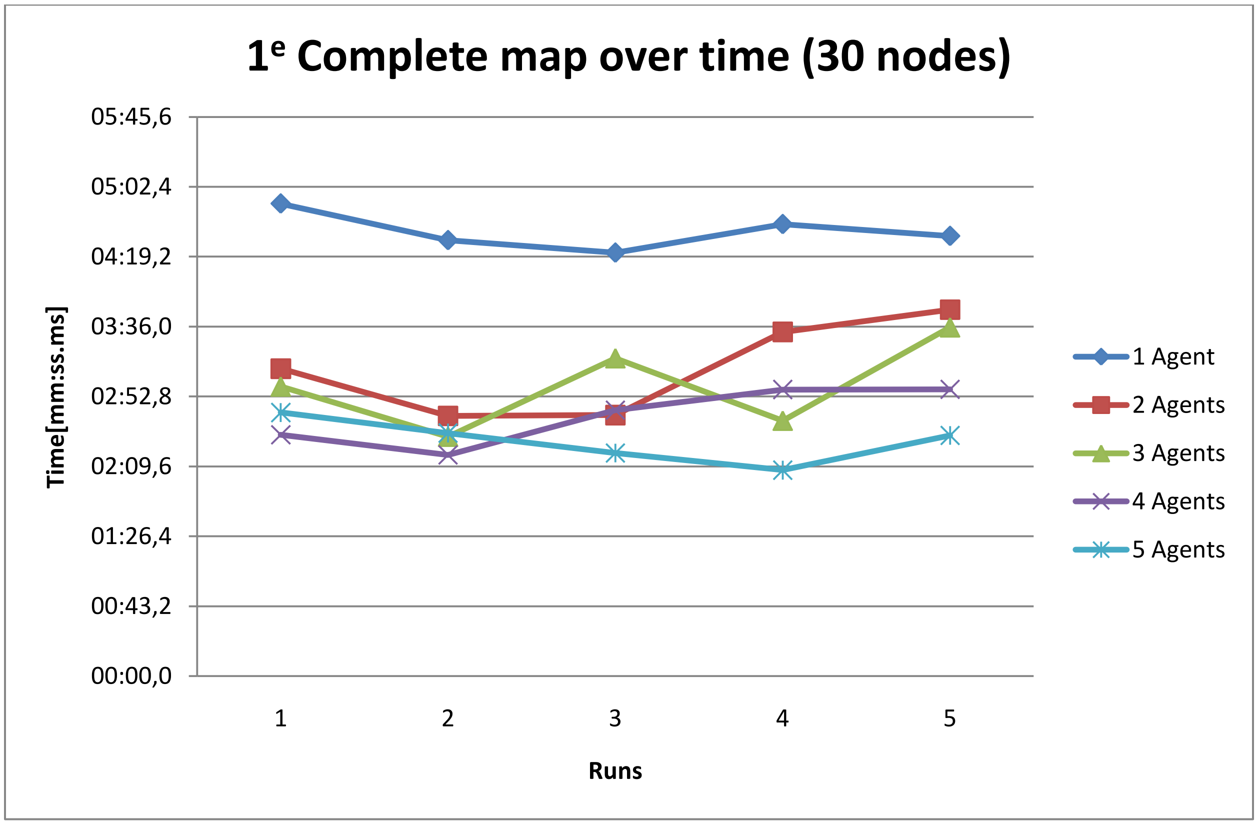


Figure 53: 30 node map results

7.4.3 The 100 node map

The 100 node map has even better results than the 30 node map if the comparison between the single agent and the five agent exploration is done. For the single agent the best result is a time of twelve minute twelve seconds and four milliseconds (12:12.4), and for the five agents the best result is five minutes and four milliseconds (5:00.4). This is a difference of seven minutes and twelve seconds (7: 12), and makes for a 118% efficiency increase.

Table 9: Data from the 100 node map

Map	Agent(s)	Runs	1e Complete	All Complete
100 nodes	1	1e	13:05.4	13:05.4
100 nodes	1	2e	12:40.4	12:40.4
100 nodes	1	3e	12:12.4	12:13.4
100 nodes	1	4e	13:00.0	13:00.0
100 nodes	1	5e	12:54.9	12:54.9
100 nodes	2	1e	10:57.2	10:59.4
100 nodes	2	2e	7:28.1	7:50.3
100 nodes	2	3e	7:40.6	10:15.1
100 nodes	2	4e	7:54.3	7:55.9
100 nodes	2	5e	9:40.7	9:43.2
100 nodes	3	1e	5:55.0	6:20.2
100 nodes	3	2e	5:58.8	7:47.4
100 nodes	3	3e	8:02.5	8:20.9
100 nodes	3	4e	5:57.4	6:01.5
100 nodes	3	5e	7:07.1	7:11.9
100 nodes	4	1e	5:45.9	6:11.5
100 nodes	4	2e	7:54.6	8:28.5
100 nodes	4	3e	6:25.1	6:31.5
100 nodes	4	4e	5:38.5	7:20.8
100 nodes	4	5e	6:31.5	6:59.8
100 nodes	5	1e	5:45.5	6:18.8
100 nodes	5	2e	5:26.4	5:47.5
100 nodes	5	3e	6:31.9	7:07.4
100 nodes	5	4e	5:43.2	5:58.1
100 nodes	5	5e	5:00.4	5:46.4

However it does contain a little inefficient behavior of the exploring agents. The fact that the map is this size and the maximum amount of agents is only five entails that the situation can occur that all the agents have the same incomplete map and tend to explore along the same route. This can result in grouping the agents together, this is very inefficient because the agents do not exchange new information anymore. This also explains the peaks in the graph with three and four agents, where the rest of the results are consistent.

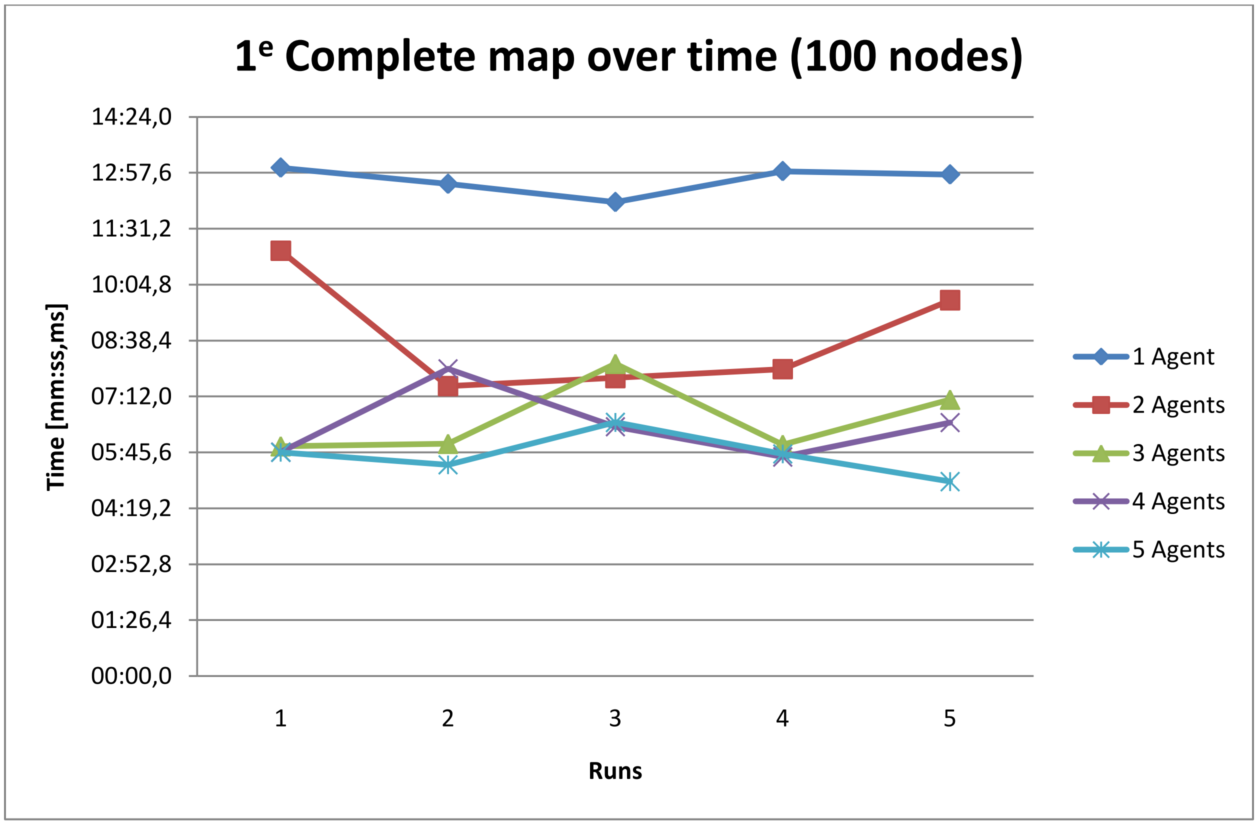


Figure 54: 100 node map results

7.5 GUI usability

The GUI usability test is used to test the GUI on performance, usability and learn-ability. For the GUI usability test, multiple test users were used to test all the before mentioned properties. A total of 7 users did the GUI usability test by performing a list of tasks and answering a couple of follow-up questions (the tasks list and the questionnaire can be viewed in Appendix A and Appendix B). The test users were first instructed briefly on the purpose and the idea behind the DMCA, and after the short explanation they did the task list (Appendix A) and the questionnaire (Appendix B).

All the test users have different back grounds, ages and experience but all have an acceptable level of knowledge and experiences with computers and different applications. Naturally 7 test users is not enough to do a full or complete usability test but it can be used to test the GUI and find the biggest problem area's or unclear situations. In Table 10 the conclusions which are based on the answers on the questionnaire are displayed.

Table 10: Results from questionnaire

Results
The main simulation window was easy to use and learn.
Disabling buttons is a good method of guiding the user.
Most problems with the map creator
Zooming problems
If a personal view window is open and that specific pda is removed from the simulation, the window stays active but it's content is removed.
For some icons it is not clear what they do
When creating or adapting a map, sometimes aligning chooses the wrong alignment point
Sometimes editing maps creates errors in the map which results in invalid maps

8 Conclusions and future work

The conclusions which can be made from this research project are divided in three difference categories. The project build up, project conclusions and future work.

8.1 Project build up

Short answers for the project build up will be provided, which can be used to generate a generic overview of the project and the encountered theories, issues and decisions.

Literature study

To summarize the literature study it can be concluded that the general subject of graph construction is studies a lot, and mainly focuses on the single agent point of view. When using single or multiple agents the tried and true method is usually to use labels to create a labeled graph. Other technique which are used are the leader election method and the fact that a graph has a tree design or can be reconstructed as a tree.

Model and design

The proposed method to solve the problem definition is to share information on the anonymous graph between multiple exploring agents and to construct the entire graph from this gathered information. This approach uses the maximal common sub graph problem to determine if multiple sub graphs can be fused together to eventually create the entire graph. By constantly sharing new additions and information with all the involved agents the construction of the entire graph will have to be realized more efficiently and in less time than it will take a single agent to construct the entire map.

Implementation

Implementing the model design was done with the help of the Distributed Map Construction Application (DMCA). This application provided a simulated environment in which a variation of maps can be created and used to test the design with a diverse amount of exploring agents. The application was created with the programming language C# and uses the XML format to store its maps in.

Tests

The tests were conducted with the help of three different maps which could contained between one and five agent(s) at one time. The three maps distinguish themselves by having different amounts of nodes. A map with ten nodes, a map with thirty nodes and a map with a hundred nodes were used. For every map twenty five simulations were executed; five with one agent, five with two agent and so on. The shortest time in which an agent had constructed the entire map was used as the result for one simulation.

8.2 Project conclusions

For the project some conclusions can be made in relation to the research of distributed map constructions. The distribution of the exploration task is a good way to increase the efficiency of the map construction. The results shown in chapter 7 certainly display a significant gain in constructing an unknown and anonymous graph when using this approach.

From these results the assumption can be made that the combination of larger maps and a high amount of agents have a positive effect on the algorithm. This shows a clear correlation between the size of the map and the amount of agents.

Another positive influence can occur when a node in the graph provides a lot of information. The most information a node contains the more likely it is that the algorithm will find a correct match within a short time. For example if a node has a tag and one or multiple connected nodes or edges have a tag this is a very unique node and can be a good starting point from which to start of the algorithm.

However there are still some issues which counteract the improvement of the results. Some of the issues which must be addressed are:

- Grouping;
- Acceptance level;
- Deviated agents;
- Technical.

First of all the grouping issue, already covered in the paragraph 7.4.3 it is the situation in which the agents group together and travel along the same route. This issue occurs because all the agents have the same exploration behavior and have a relative small communication range. The small communication range is an issue because the agents are already in close proximity of each other and have the tendency to group together.

The acceptance level issue only plays a part in the smaller graph. The implemented algorithm uses a minimum of four consecutive complete nodes to establish a correct match and if the graph isn't that larger this is can be a high percentage of the entire graph. An adjustment of the acceptance level before hand can't be done because the initial map is unknown to any of the agents.

In some simulations one or two agents deviate from the whole and does not have any information exchange with the other agents. This doesn't contribute to the approach and only has a negative influence on the results.

There is a technical issue with the implementation but it's not clear where the problem originated from. When a simulation is executed in a map with more than a hundred nodes it tends to pause for half a second when executing the match and merging algorithm. But it only occurs when the agents have about a hundred nodes in their personal map. This could be a memory or a calculation issue but when all the agents perform their own calculations and use their own memory this will not be a problem anymore.

The used approach can clearly help build a complete state and view of the unknown environment. By sharing this information it is possible to plan and/or execute tasks which can help casualties or emergent situations. On a related note, distributing the knowledge is a save way to store the information in an emergent situation. If one information source is removed from the field all or most of the information is preserved by all the other sources. This can be seen as an extra advantage for distributed information sharing instead of centralized information sharing.

In conclusion the used approach and algorithms provide for good results in the field of distributed map construction. But naturally there will always be issues which will have to be improved or solved, to further increase the performance. Some improvements will be displayed in the future work paragraph.

8.3 Future work & recommendation

If this project or a similar project will continue with this research, I would first look at the correlation between the acceptance level and the size of the graph or map for optimal results. It is unlikely that the results keep improving when more agents are added to the map, so the ideal combination will have to be looked into. This can be tricky to do however because the size of the graph is unknown to start with, but for research purposes it could be beneficial.

Furthermore all the issues which are displayed in the conclusion will have to be solved. A solution for the grouping of the agents can be to implement multiple exploration behaviors and distributing them between the agents. A solution for the deviated agent(s) is a little more difficult because this situation can occur without a clear reason. It is however only an issue if all the agents will have to have a complete view of the map. If this is not the case it does not play a large part in the entire process. The technical issue will be solved when all the calculations and memory which are necessary, are available for each separate agent or user, in a real life implementation for example.

If a real life implementation of this kind of research would be realized, I would strongly suggest working with a digital compass to help the algorithm to function better. This will provide the user with a sense of direction and this way the rotation issue is removed from the field and from the calculations. Oblique edges can also be added to the maps and the algorithm. With the use of the degrees between two edges we can match the nodes with each other.

9 Bibliography

[1] Velden, M. van. "ManetLoc A location based approach to distributed world-knowledge in mobile ad-hoc networks." 2005.

[2] Norvig, Russel &. *Artificial Intelligence a Modern Approach 2e edition.* Second edition. Pearson Education (US), 2002.

[3] Gerardo Beni, Suzanne Hackwood, and Jing Wang. *Swarm Intelligence.* 1989.

[4] S. Das, P. Flocchini, S. Kutten, A. Nayak, N. Santoro. *Map construction of unknown graphs by multiple agents.* Elsevier, 2007.

[5] K. Diks, P. Fraigniaud, E. Kranakis, A. Pelc. "Tree exploration with little memory." *Journal of Algorithms 51*, 2004.

[6] P. Fraigniaud, D. Ilcinkas. "Digraph exploration with little memory." *21st Symposium on Theoretical Aspects of Computer Science.* 2004. 246-257.

[7] M. Bender, A. Fernandez, D. Ron, A. Sahai, S. Vadhan. "The power of a pebble: Exploring and mapping directed graphs." *Proc. 30th ACM Symposium on Theory of Computing.* 1998. 269–287.

[8] G. Dudek, M. Jenkin, E. Milios, D.Wilkes. "Robotic exploration as graph construction." 1991.

[9] M. Bender, D.K. Slonim. "The power of team exploration: Two robots can learn unlabeled directed graphs." *Proc. 35th Symp. on Foundations of Computer Science.* 1994. 75-85.

[10] X. Yu, M. Yung. "Agent rendezvous: A dynamic symmetry-breaking problem." 1996.

[11] A. Dessmark, P. Fraigniaud, A. Pelc. "Deterministic rendezvous in graphs." *11th European Symposium on Algorithms.* 2003. 184-195.

[12] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro. "Rendezvous and election of mobile agents: impact of sense of direction." *10th Coll. on Structural Information and Communication complexity.* 2003. 17-32.

[13] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk. "Multiple mobile agent rendezvous in a ring." *6th Latin American Theoretical Informatics Symposium.* 2004. 599-608.

[14] E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk. "Mobile agent rendezvous in a ring,." *International*

Confension on Distributed Computing Systems. 2003. 592-599.

[15] Hyacinth S. Nwana, Divine T. Ndumu. "An Introduction to Agent Technology." 2003.

[16] Levesque. *Fundamental issues with open source software development.* 2004. http://firstmonday.org/issues/issue9_4/levesque/index.html.

[17] Wooldridge, M. *Reasoning about Rational Agents.* The MIT press, 2000.

[18] Bratman, Michael E. *Intention, Plans, and Practical Reason.* Stanford: CSLI publications, 1999.

[19] *Jade - Java Agent DEvelopment Framework.* 2008. http://www.jade.tilab.com.

[20] E. Remolina, B. Kuipers. *Towards a General Theory of Topological Maps.* University of texas, 2002.

[21] Thrun, S. *Learning metric-topological maps for indoor mobile robot navigation.* Carnegie Mellon University, 1997.

[22] H. Bunke, A. Kandel. *Mean and maximum common subgraph of two graphs.* University of Bern, University of south Florida, 2000.

[23] Valiente, G. *Subgraph isomorphism and related problems.* Technical University of Catalonia, 2001.

[24] K. Doty, S. Seed. *Autonomous agent map construction in unknown enclosed environments.* University of Florida, 1994.

[25] Z. Butler, A. Rizzi, R. Hollis. *Distributed coverage of rectilinear environments.* Carnegie Mellon University, 2001.

[26] Kurt Konolige, Jens-Steffen Gutmann, Benson Limketkai. "Distributed Map-Making." 2003.

[27] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas and Richard LaMaire. "Distributed Topology Construction of Bluetooth wireless Personal Area Networks." *IEEE Journal on selected areas in communications* volume 23 (March 2005).

[28] Lotta Danielsson-Murphy, Thaddeus Murphy. "Global Positioning Systems, A technical Assessment Paper." 1997.

[29] Srdan Capkun, Maher Hamdi, Jean Pierre Hubaux. "GPS-free positioning in mobile Ad-Hoc." 2001.

[30] Shantanu Das, Paola Flocchini, Shay Kutten, Amiya Nayak, Nicola Santoro. "Distributed Exploration of Anonymous Graphs by Multiple Agents." 2005.

[31] Edmund H. Durfee, Jeffrey S. Rosenschein. "Distributed Problem Solving and Multi-agent Systems: Comparisons and examples." 1994.

List of Figures

List of Tables

Appendix A

Graphical User Interface usability test (Task list & Questionnaire)

Opdrachten

Bij voorbaat bedankt voor het deelnemen aan deze gebruiksvriendelijkheids test. Probeer de opdrachten zo goed mogelijk uit te voeren en onthoud of maak aantekeningen van de opdrachten die moeilijk waren of die niet gelukt zijn.

Na het vervullen van alle opdrachten volgt er een korte vragenlijst

Opdrachten:

1. Open een nieuwe simulatie map;
2. Voeg een Pda toe aan de simulatie;

3. Voeg twee Pda's aan de simulatie toe;
4. Verwijder all pda's;

5. Voeg nog een keer twee pda's toe;
6. Verander de simulatie opties;

7. Pauzeer de simulatie;
8. Hervat the simulatie;

9. Bekijk een pdas personal view;
10. Pauzeer een individuele pda;
11. Hervat de pda;

12. Creëer een nieuwe map;
    a. Voeg een aantal nodes toe;
    b. Verbind ook bestaande nodes aan elkaar;
    c. Voeg aan deze map een blokkade toe;
    d. Verwijder een of meerdere nodes;

13. Open uw eigen gecreëerde map en start een simulatie;

Nogmaals bedankt voor u deelnamen en vergeet niet uw vragenlijst in te vullen en in te leveren.

Met vriendelijke groet,

P. Waasdorp

Appendix B

Questionnaire

Vragenlijst

Hoe voldeed het programma aan uw verwachtingen?

Heel goed	Goed	Normaal	Slecht	Heel slecht
●	○	○	○	○

Hoe ervaarde u de dit programma?

Heel goed	Goed	Normaal	Slecht	Heel slecht
○	○	○	○	○

Hoe ervaarde u de simulatie omgeving?

Heel goed	Goed	Normaal	Slecht	Heel slecht
○	○	○	○	○

Was het duidelijk wat het programma moest doen (omcirkel wat van toepassing is)?
Ja / nee

Zaten er moeilijke opdrachten tussen en zo ja welke (opdracht nummer is genoeg) en waarom?

Opdrachtnummer:……………………………………………………………………………………………………

Waarom:………
………

Zijn er elementen die opvallen en zo ja welke (negatief of positief)?
………
………

Heeft u nog suggesties om het programma te verbeteren?
………
………