Dissertation

Project Title: Distributed Agents Location Discovery on the Ring

Student:
Thomas Gorry

Supervisors:
Prof. Leszek Gąsieniec
Dr. Russell Martin

September 20, 2011
## Contents

1 Abstract 3

2 Introduction and Background 3
   2.1 Problem 3
   2.2 Model 3
   2.3 Approach 4
       2.3.1 Overview 4
       2.3.2 Project Lemmas 4
       2.3.3 Software 8

3 Design - Simulation Software 9
   3.1 Overview 9
   3.2 Interface Design 9
   3.3 UML Diagram 12
   3.4 Pseudo-Code 13
       3.4.1 paintComponent - GUI class 13
       3.4.2 run - Move class 13
       3.4.3 Step - Agents class 14
       3.4.4 move_Di1j0_Si1j1 - Agents class 15

4 Design - Algorithm 15
   4.1 Overview 15
   4.2 Pseudo-Code - Discovery Algorithm 16

5 Realisation 17
   5.1 Software 17
       5.1.1 Overview 17
       5.1.2 Positions on the Ring 18
       5.1.3 Analysis Functions 18
       5.1.4 Problems Occurred 19
   5.2 Algorithm 19

6 Evaluation 22
   6.1 Testing 22
       6.1.1 Rotational Shift 22
       6.1.2 Odd n 22
       6.1.3 Variable Speeds 25
   6.2 Strengths 29
   6.3 Weaknesses 29

7 Professional Issues 29
   7.1 Code of Conduct 29
       7.1.1 Public Interest 29
       7.1.2 Professional Competence and Integrity 30
       7.1.3 Duty to Relevant Authority 30
       7.1.4 Duty to the Profession 30
   7.2 Code of Good Practice 30
## 8 Conclusions

8.1 Project Outcome ................................................................. 30
8.2 Further Work ............................................................... 31

## 9 Bibliography

## 10 Appendices

10.1 Appendix A - Original Specification Documentation ................................. 33
10.2 Appendix B - Original Design Documentation and Slides ........................... 39
10.3 Appendix C - Original Final Presentation Documentation and Slides .............. 55
10.4 Appendix C - Original Time Plan ............................................ 67

### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Periodicity Diagram</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Rotation Index Diagram</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Home Screen Design</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Speed Setting Input Box Design</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Main Screen Design</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Screenshot Menu Design</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Positional Data Menu Design</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>UML Class Diagram Design</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>Home Screen Realisation</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>Main Screen Realisation</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>Positional Data Menu Realisation</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Screenshot Menu Realisation</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>Positional Data Test 1</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Screenshot Data Test 1</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>Screenshot Data Test 1</td>
<td>21</td>
</tr>
<tr>
<td>16</td>
<td>Screenshot Data Test 1</td>
<td>21</td>
</tr>
<tr>
<td>17</td>
<td>Screenshot Data Test 1</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>Positional Data Test 2</td>
<td>22</td>
</tr>
<tr>
<td>19</td>
<td>Screenshot Data Test 3</td>
<td>23</td>
</tr>
<tr>
<td>20</td>
<td>Screenshot Data Test 3</td>
<td>23</td>
</tr>
<tr>
<td>21</td>
<td>Screenshot Data Test 3</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>Screenshot Data Test 3</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>Screenshot Data Test 4</td>
<td>25</td>
</tr>
<tr>
<td>24</td>
<td>Screenshot Data Test 5</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>Screenshot Data Test 5</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>Screenshot Data Test 5</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>Screenshot Data Test 5</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>Screenshot Data Test 5</td>
<td>27</td>
</tr>
<tr>
<td>29</td>
<td>Screenshot Data Test 5</td>
<td>27</td>
</tr>
<tr>
<td>30</td>
<td>Screenshot Data Test 5</td>
<td>28</td>
</tr>
<tr>
<td>31</td>
<td>Positional Data Test 5</td>
<td>28</td>
</tr>
</tbody>
</table>
1 Abstract

This project proposes a model of \( n \) uniform anonymous agents patrolling on a ring. The agents share a common sense of direction and have a maximum speed of 1. These agents have no communication abilities although they can sense that they have collided with another agent and upon collision will immediately turn and travel in the opposite direction they were traveling along the ring. The agents are able to measure distance travelled and have access to a pool of randomised bits. Also the agents perform their actions in synchronised rounds. The problem of patrolling this ring by the anonymous agents in an efficient way required that each agent must first become aware of the starting locations of each of the other agents on the ring. This project investigates how this is possible in the model describes with no communication. In order to help give understanding and insight into the mechanics of the problem this project also required the production of software that could simulate these agents on the ring. Investigation into this problem shows that it is indeed possible to achieve location discovery for each agent in this model. The reasons for this is that the movement of these agents on the ring can be split up into blocks of time such that at the end of each block of time each agent is currently on the starting location of one of the agents on the ring. The project describes an algorithm that can achieve this in time of \( O(n \log^2 n) \) for both the case when the number of agents on the ring is odd and also when the number of agents is even. Further investigation in this problem could be down several avenues. For example the question of can this location discovery and subsequent organisation and patrolling be done when the agents are not aware of \( n \). Also the software that has been produced for this project can provide some insight into the case where the agents are able to travel at different maximum speeds.

2 Introduction and Background

2.1 Problem

This project is a research based project that focuses on the area of agent coordination and organisation, specifically that of coordination and organisation on a ring. This problem has its roots in the Mobile Agent Fence Patrolling Problem [3] and the project looks at the behaviour of the agents on a ring to propose a possible organisation solution in the model explained below. This area of research has its roots in the patrolling problem that has been studied by many in robotics [3] and has been looked into in the form of patrolling in a closed area by [8]. Network Patrolling is defined by [2] as a perpetual process performed in a static or in a dynamically changing environment. There have been different methods to measure the success of this in previous work. For example, in [8] they study patrolling in adversarial environments, in which the robots’ goal is to maximize their rewards. These rewards are received if the robots manage to observe the adversary, which tries to evade the patrolling robots and the more rewards the better the measurement of optimisation.

2.2 Model

Consider the following model of a ring with length 1 and \( n \) anonymous agents situated randomly along its circumference with each position unknown to other agents. All agents are allowed to walk along the ring in any direction (Clockwise, \( C \), or Anti-Clockwise, \( A \)) where there is a universal notion of direction and time \( t \). The agents on this ring perform actions in synchronised rounds. At the beginning of each round an agent randomly chooses a direction to traverse the ring, either clockwise or anti-clockwise. Each Agent has the abilities of measuring distance, is aware of \( n \), and is also collision aware. Furthermore each Agent has access to its own set of randomised bits that it can access if decisions need to be made. All Agents have the limitation of not being able to communicate with each other and are unable to leave marks on the ring and also have zero visibility. However, on the conclusion of each round the agent receives trajectory information of its movement during the round; this can then be stored by the agent for further analysis later on. We assume that while the set of \( n \) Agents are moving along the ring they have a maximal speed of 1 and that
in $t = 1$ an Agent using this maximal speed makes exactly one complete cycle on the ring. Where Agents collide they change direction (instantaneously) and travel this new direction around the ring. Agents are able to move between 0 and their maximal speed and are able to choose to change direction or stop for some amount of time. The use of the word ring in throughout this program will be used to represent a circle, in this way it is easier to model the problem for our purposes.

2.3 Approach

2.3.1 Overview

For the purpose of this project algorithmic efficiency was measured by its capacity to optimise the frequency of visits to each point of the ring, known as Idle Time. This is the general approach taken in this field [2, 3]. For patrolling of a unidirectional circle with constant equal speed entities it is widely conjectured that the most efficient method, in terms of minimising Idle Time, is to arrange these patrolling entities at equal distances around the network and have them walk in a constant unified direction [2]. Therefore the Cyclic-Strategy of patrolling is the one that we have chosen for this project. This strategy is described by [1] and studied in [2]. There has been research in other directions, [4], that look towards the natural world to come up with a solution to the patrolling problem and have used the idea employed by ants of pheromone stamping and then having a "leader" Agent be responsible for finding the shortest path before employing a spreading algorithm for each Agent to separate to equal distances while following this pheromone trail. Unlike previous work in this area, that has included elements of communication between agents, swarm intelligence, distinct number of agents and so on, our research assumes a much simpler model and looks towards breaking the problem up into two phases. Firstly each agent locating the original starting position of every other agent and then entering a synchronisation phase before commencing the actual patrolling of the network. The synchronisation phase of this project is similar to that of the Firing Squad Problem proposed by J. Myhill in 1957 [7] and it is likely that an answer lies in the study of this problem and its solutions.

2.3.2 Project Lemmas

While researching this problem we observed a number of things that would later become very useful to us in formulating an approach to this problem. We write these observations as lemmas in the following way:

**Lemma 1.** Periodicity At the start of each round baton $b_i$ resides at position $p_i$, for all $i = 0, \ldots, n - 1$.

*Proof.* Since in the beginning of each round baton $b_i$ resides at $p_i$ and throughout the round it keeps moving in the same direction with the speed one, $b_i$ must arrive at $p_i$ on the conclusion of this round. This can be seen in the diagram below. 

---

1This section is based heavily on the work currently being carried out in [5] by T. Friedetzky, L. Gąsieniec, R. Martin and T. Gorry
Note that a virtual baton is always carried by some agent. Thus using Lemma 1 one can conclude that at the start of each round the agents populate initial locations $p_0$ through $p_{n-1}$. In fact, one can state a more accurate lemma.

**Lemma 2. Rotation Index** At the start of each round, for all $i = 0, \ldots, n-1$, agent $a_i$ resides at position $p_{i+k}$, for some $k \in \{0, \ldots, n-1\}$.

*Proof.* Note that all initial positions are populated by the agents. The agents never overpass. I.e., agent $a_i$ has always the same neighbours $a_{i-1}$ and $a_{i+1}$. Thus if $a_i$ resides at position $p_{i+k}$, $a_{i-1}$ and $a_{i+1}$ must reside at the respective locations $p_{i+k-1}$ and $p_{i+k+1}$.

Using observation from Lemma 2, consider respective locations $p_{i+k_1}$ and $p_{i+k_2}$ of agent $a_i$ at the start of two consecutive rounds. One can conclude that during one round all agents rotated along initial positions by a rotation index of $r = k_2 - k_1$. How this relates to the agents on the ring can be observed in Figure 2.
Lemma 3. During one stage the rotation index \( r \) remains unchanged.

Proof. The movement direction of each agent is determined by the constant movement direction of the virtual button possessed by the agent. Since at the beginning of each round virtual batons reside at their original positions and they do not change their directions, the rotation index throughout each stage must remain unchanged.

We show now that the rotation index \( r \) depends on the initial choice of random directions adopted by the agents. Consider the first round of any stage. Let sets \( B_C \) and \( B_A \) contain virtual batons that move during this round in the clockwise and in the anticlockwise directions respectively, where \( |B_C| = n_c, |B_A| = n_a \), and \( n_c + n_a = n \). We say that during this stage virtual batons form a \((n_c, n_a)\)-configuration.

Lemma 4. In a stage with \((n_c, n_a)\)-configuration the rotation index \( r = n_c - n_a \).

Proof. It is enough to prove the thesis of the lemma for one agent since all other agents will rotate their positions accordingly. W.l.o.g., assume that baton \( b_i \) is in \( C \). Recall that at the beginning of any round baton \( b_i \) is aligned with position \( p_i \) (original location of \( a_i \)) and assume that at the beginning of the considered round \( b_i \) is carried by agent \( a_j \). First note that \( b_i \) can be only exchanged with batons from \( B_A \) since batons from \( B_C \) move with the same speed in the clockwise direction. Moreover during any round every baton from \( C \) is exchanged with every baton from \( A \) exactly twice at certain antipodal points of the ring. Thus during any round baton \( b_i \) is exchanged between colliding agents exactly \( 2n_a \) times. Also since \( b_i \) moves in the clockwise direction during each exchange an index of the new hosting agent is increased by one. Thus at the end of the considered round when \( b_i \) arrives at \( p_i \) it is hosted by agent \( a_j + 2n_a \). This leads to conclusion that the rotation factor \( r = -2n_a \). Note that focusing on batons from \( A \) one can use an analogous argument to prove that the rotation factor \( r = 2n_c \). Now since \( n_c + n_a = n \) we get \( -n_a = n_c \pmod{n} \) and finally \( -2n_a = 2n_c \pmod{n} \) admitting the uniform across all agent rotation index \( r \). Finally, we add \( n_a + n_c \) (that has value 0 modulo \( n \)) to \(-2n_a\) and we obtain the rotation index \( r = n_c - n_a \).

Lemma 5. For any odd \( n > 0 \), with high probability a successful stage occurs within the first \( O(\log^2 n) \) stages.

Proof. We already know that we target a distribution of directions, s.t., \((n_c - n_a, n) = 1\). To simplify our task we focus only on prime values of \(|n_c - n_a| < \sqrt{n}\). Note that the probability that during a stage the value \(|n_c - n_a|\) is obtained is

\[
\frac{2 \cdot \binom{n}{n_c}}{2^n}.
\]

Using Stirling’s factorial approximation one can prove that this probability is \(\Omega\left(\frac{1}{\sqrt{n}}\right)\), for all \(|n_c - n_a| < \sqrt{n}\). We also know [6] that for an integer \(m\) large enough (\(> 15,985\)) the \(m\)-th prime number is not larger than \(n(\log n + \log \log n)\). This can be also interpreted that for \(m\) large enough there are \(\Omega\left(\frac{m}{\log m}\right)\) prime numbers smaller than \(m\). In particular, we can conclude that there are \(\Omega\left(\frac{\sqrt{n}}{\log n}\right)\) primes between \(0\) and \(\sqrt{n}\). Thus the probability that each stage is successful with probability \(\Omega\left(\frac{\sqrt{n}}{\log n} \cdot \frac{1}{\sqrt{n}}\right) = \Omega\left(\frac{1}{\log n}\right)\). Finally, one can observe that after \(O(\log n)\) stages the probability of reaching a successful stage is constant, and after \(O(\log^2 n)\) stages the probability is very high.

Lemma 6. For any odd \(n\), the number of rounds required to perform full discovery of agents initial positions is bounded by \(O(n \log^2 n)\) with very high probability.

Proof. Each stage is formed of at most \(n\) rounds, and with high probability the algorithm accomplishes the discovery task in \(O(\log^2 n)\) stages.

Lemma 7. For any even \(n > 0\), each agents learn the positions of the others within the first \(O(\log^2 n)\) stages with high probability.
Proof. In order to simplify the proof we focus on two sets of pairs of positions: \( P_0 = \{(p_{2j}, p_{2j+1}) : j = 0, 1, \ldots, \frac{n}{2}\} \) and \( P_1 = \{(p_{2j+1}, p_{2j+2}) : j = 0, 1, \ldots, \frac{n}{2}\} \). Within every set each pair contains distinct agent’s initial positions and every such a position belongs to some pair. We split consecutive successful stages (with \((n_c - n_a, n) = 2\)) of procedure Discovery to two alternating sequences \( S_0 \) and \( S_1 \) where in stages from \( S_0 \) we consider pairs from \( P_0 \) and in stages from \( S_1 \) we consider pairs from \( P_1 \). W.l.o.g., consider sequence \( S_0 \). Recall that in these stages every agent visits every other initial position on the ring. Thus if, e.g., in the beginning of the first round of this stage agent \( a_{2j} \) moves to direction \( C \) and agent \( a_{2j+1} \) moves to direction \( A \), after the first stage these two agents learn their relative positions, and in the remaining \( \frac{n}{2} - 1 \) rounds of this stage all other agents in \( P_0 \) learn \( p_{2j+1} \) as well as all other agents in \( P_1 \) learn \( p_{2j} \). Thus we need to consider enough number of stages in \( S_0 \), s.t., for each \( j = 0, \ldots, \frac{n}{2} \) there is a stage in which agent \( a_{2j} \) starts moving towards direction \( C \) and agent \( a_{2j+1} \) starts moving towards direction \( A \). During each such stage, we randomly populate \( \frac{n}{2} \) pairs in \( P_0 \) with pairs of directions: \((A, A), (A, C), (C, A), \text{ and } (C, C)\), under the assumption that \(|n_c - n_a| < \sqrt{n}\). Since we are interested only in pair \((C, A)\), we first estimate from below the expected number of \((C, A)\)s generated during each successful stage in \( S_0 \). We generate pairs sequentially in random assuming that initially the number of As and Cs is at least \( \frac{n}{2} - \sqrt{n} \). And we generate these pairs until either the remaining number of As or the remaining number of Cs is smaller than \( \frac{n}{4} - \sqrt{n} \). This means that we generate at least \( \frac{n}{2} = \frac{n}{8} \) pairs. Since during this time the number of As and Cs is fairly balanced one can compute that the probability of picking a mixed pair \((C, A)\) (or \((A, C)\)) is at least \( \frac{1}{5} \).

Thus the expected number \( \mu \) of \((C, A)\)s generated is at least \( \frac{n}{8} \cdot \frac{1}{5} = \frac{n}{40} \). Using the lower tail in Chernoff bounds \( \| \) one can estimate that we generate less than \( \frac{n}{50} \) pairs (multiplicative parameter \( \delta = \frac{1}{5} \)) of the form \((C, A)\) with probability smaller than \( e^{-\frac{n}{50}\left(\frac{1}{2}\right)^2} < \frac{1}{2}\), for \( n \) large enough. We will say that a stage in \( S_0 \) is valid if at least \( \frac{n}{2} \) \((C, A)\)s are generated during this stage. Each stage in \( S_0 \) is valid with probability at least \( \frac{1}{2} \) and during each valid stage at least \( \frac{n}{50} \) \((C, A)\)s are randomly distributed among pairs in \( P_0 \). Recall the Coupon Collector Problem (CCP) in which one player must collect \( m \) coupons. During each attempt the player receives each coupon with probability \( \frac{1}{m} \). It is well known that the player will collect all coupons after \( O(m \log m) \) attempts (i.e., after \( c \cdot m \log m \) attempts, for some constant \( c \)) with very high probability. Note that CCP can be also executed in consecutive stages, where each stage can be formed of, e.g., \( \frac{n}{50} \), consecutive attempts. In this case one can conclude that it is enough to run \( \frac{50c \cdot m \log m}{n} \) stages to collect all coupons with very high probability. Note that generation and further distribution of \((C, A)\)s in valid stages can be seen as a "more powerful" version of coupon collection run in (respectively in size) stages. During a valid stage at least \( \frac{n}{50} \) sequential attempts (generated \((C, A)\) pairs) are made. Each attempt we draw one out of \( m = \frac{n}{2} \) coupons (positions in \( P_0 \)). Compare a single stage in CCP with a valid stage in procedure Discovery. Note that if in CCP a specific coupon is drawn more than once respective second and further attempts are void. In other words, these attempts are wasted. In a valid stage, however, if an attempt results in a coupon (position in \( P_0 \) that has been already collected in this stage, the attempt is repeated until not yet collected coupon is found. Thus in each valid stage we may in fact generate more (but definitely not less) coupons comparing to the respective stage in CCP. Thus the number of valid stages required to collect all \( m = \frac{n}{2} \) coupons (positions) will not exceed the number of stages with \( \frac{n}{50} \) attempts in CCP. Furthermore since the number of such stages in CCP is \( \frac{50c \cdot m \log m}{n} \) where \( m = \frac{n}{2} \) we obtain the upper bound \( 100c \log n \) on the number of valid stages required by procedure Discovery. In other words, if Discovery enters \( 100c \log n \) valid stages the agents discover the initial positions of others with very high probability. Since also in this case the probability that a stage is valid is \( \Omega(\frac{1}{\log n}) \) one can show using lower tail in Chernoff bound that \( O(\log^2 n) \) rounds should suffice to generate \( 100c \log n \) valid stages with high probability. And since these stages suffice to generate all \((C, A)\) pairs with high probability \( O(\log n) \) stages of procedure Discovery suffice in this case too.

\[ \square \]

**Theorem 8.** For any \( n > 0 \), large enough, each agents learn the positions of the others within the first \( O(\log^2 n) \) stages with high probability.
2.3.3 Software

For the purpose of this project software was developed to aid our understanding of the problem and provide visual aid for observing the lemmas stated above in real time. It was agreed that this software was to have the following features:

- The user should be able to visualise in real time what is happening on the ring and to each agent.
- The user should be able to specify how many agents are present on the ring.
- The user should be able to have an option to specify a custom speed for each agent.
- The user should be able to have an option to employ acceleration on the Agents such that after collisions the Agents take some time to achieve maximum speed again.
- There should also be a function to show previous time periods in the animation.

Also it was discussed that more comprehensive functions could be added to the software in the future in the event of the project scope shifting directions to include new spheres of interest, this is due to the fact that I will be continuing study in this, and surrounding areas, for a PhD starting in October 2011. Also it should be mentioned that due to the difficulty of simulating a continuous setting a discretized version of the ring was used for the software. This will mean that agents may only occupy integer positions along the circumference of the ring rather as opposed to the continuous ring being explored by the theoretical side of this project in which all positions on the ring are accessible.
3 Design - Simulation Software

3.1 Overview

For the purpose of this project software was developed to aid our understanding of the problem and provide visual aid for observing the lemmas stated above in real time. It was agreed that this software was to have the following features:

- The user should be able to visualise in real time what is happening on the ring and to each agent.
- The user should be able to specify how many agents are present on the ring.
- The user should be able to have an option to specify a custom speed for each agent.
- There should also be a function to show previous time periods in the animation.

Also it was discussed that more comprehensive functions could be added to the software in the future in the event of the project scope shifting directions to include new spheres of interest, this is due to the fact that I will be continuing study in this, and surrounding areas, for a PhD starting in October 2011. Also it should be mentioned that due to the difficulty of simulating a continuous setting a discretized version of the ring was used for the software. This will mean that agents may only occupy integer positions along the circumference of the ring rather as opposed to the continuous ring being explored by the theoretical side of this project in which all positions on the ring are accessible.

3.2 Interface Design

![Figure 3: Home Screen Design](image)

Shown above is a screenshot of what the home screen was intended to look like when the software was finished. From here the user will be able to specify the number of agents to be included in the simulation as well as decide whether or not they wish to manually change the speeds of each agent. The feature of including agents of different speeds will be incorporated with a view for further research into this area as this project is primarily concerned with agents of the same speed and in a model where acceleration does not take place. The acceleration feature has been removed from the criteria since the original design document, which can be found in the Design appendix section of this document, as it was too complex to be implemented and as it only applied to further research in this area it was deemed acceptable to omit this feature from the current version of the software. However, inclusion of the acceleration feature may occur if future research goes in this direction.
Above is a screenshot of what the user will be asked if they choose the option of setting the speed of each agent.

![Speed Setting Input Box Design](image1)

**Figure 4: Speed Setting Input Box Design**

Above is a screenshot of the main screen that the user will be shown when simulating the model. From here the user will be able to see an animation of what the ring environment looks like as the agents move along its circumference and change direction upon collision. It is hoped that this main screen will provide a way for possible patterns to be recognised and a visual testing feature for the solution to the Network Patrolling Problem described above. From this screen the users will be able to tell what Stage and Round the agents are currently in. The user will also be able to choose to Start or Stop the system or start a new simulation, clicking this button will take the user back to the home screen where they will be able to set up a new scenario. There is also buttons that the user can click on to view the screenshot menu, where still images of the ring when at the end of each round are stored and can be viewed for analysis, and the Position Data menu, where positional data for all agents on the ring at the end of each round can be viewed.

![Main Screen Design](image2)

**Figure 5: Main Screen Design**
Shown above is a screenshot of the screenshot menu that the user will be shown when simulating the model. Clicking on any of the images on the right of the screen will show them in a large view in the main window. For each collision a still image of the ring will be taken and stored in this menu for further viewing and analysis. Therefore the thumbnail menu of screenshots will be dynamic in that it continuously changes size when a simulation is running; this is why it will be contained in a scrollable window. The user can use the button at the bottom of the menu panel to navigate back to the main animation screen.

Shown above is a screenshot of the positional data menu that the user will be shown after selecting this option from the main animation window. The data here is hoped to help provide proof of the lemmas mentioned earlier and provide a clear understanding of how the agents move around the ring. The data is shown as text that can resize itself dynamically so that each line for the rounds will never take up more than one row, regardless of the number of agents. This allows the user to see a clear view in columns of the agents’ positions at the end of different rounds and compare them as such. This is also set up as a scrollable window so the number of rounds that can be viewed here is not limited. The user can use the button at the bottom of the menu panel to navigate back to the main animation screen.
3.3 UML Diagram

Here is the current version of the UML diagram for this project. The original copy can be found in the Design appendix section of this document. If required an online version of this class diagram can be found at http://www.csc.liv.ac.uk/~m0tg/MSc.html

Figure 8: UML Class Diagram Design
3.4 Pseudo-Code

For the purpose of this design document Pseudo-code and descriptions for interesting or particularly challenging methods will be included for the simulation software code. The methods that will be displayed here will include the `paintComponent` method from the GUI class; the `run` method from the Move class; the `step` method and an example of a movement check method from the Agents class.

3.4.1 `paintComponent` - GUI class

The `paintComponent` method is used to construct the ring that the agents will be traveling along as well as the agents themselves. In order for the system to work accurately the positions for each of the agents must also be accurate on the ring. Therefore for this reason the positions of the agents are worked out using the Cosine rule and general Trigonometry. Working out the positions of the agents on the ring in this way means that we can effectively label each point that can be visited on the ring circumference with the degree of angle the radius to that point would create between itself and the x-axis running through the centre point of the circle with the centre being the origin of (0,0). This way each point along the circumference of the ring can be represented by an integer between 0 and 359 (as 360 would be the same as position 0). The agents can then move along the ring in a similar fashion by simply changing their degree position by 1 each time, add 1 to go clockwise around the ring and minus 1 to go Anti-Clockwise. This refers to the discretized setting discussed earlier. However, this means that if the number of agents does not divide 360 then \( \frac{1}{n} \) placement of agents cannot be accomplished. Therefore some further thought must go into dealing with this issue.

```java
ringColor = white
getcenterCoordinateX()
getceterCoordinateY()
getRadius()
drawOval(centerCoordinateX, centerCoordinateY, radius * 2, radius * 2)
for i < number of Agents do
    positionOnTheRing = (radius * (cos(2 * pi * (theta[i] / 360))))
    AgentColor = Colors[i]
    i ← i + 1
end for
```

3.4.2 `run` - Move class

When the user clicks on the "Start" button in the GUI then this method will be called as a new thread will be created to simulate the movement of the agents. This method will be responsible for updating the ring and as such the animation of the agents traveling along it, this is controlled by the loop in the method that will keep running until the user pauses it or terminated it. The method that handles retrieving the image of the ring at the end of each round is also called in this method at the end of each round.

```java
terminate = FALSE
GUI.currentRound = 0
loop
    check = (GUI.currentStep / 360)
    if check == check then
        GL.ScreenCapture()
        newround = TRUE
    else
        newround = FALSE
```
end if
Agents.step()

 GUI.currentStep ← GUI.currentStep + 1
if newround == TRUE then
   Update Round Number
end if
Update contentPanel and ringDisplay
Tell Thread to sleep for 20 milliseconds
if terminates == true then
   return
end if
end loop

3.4.3 Step - Agents class

This method is called from the run method in the Move class. It essentially governs each step that the agent takes while walking around the ring. Firstly it loads in the current positions and directions of each agent on the ring. Then for each agent it checks a number of things to decide on where the agent will move to in the next step. The method does this by firstly checking the current direction of the agent and based on this it calls either the move_Di1 method or the move_Di0 method. Once these methods have run then it calls the update method. Both of the two methods mentioned first check to see what the speed of the agent, $i$, is and also for every other agent on the ring, $j$, they check to see if $j$ is traveling the same direction as $i$ or not. Based on these factors the methods will choose what sort of collision checks need to be made between $i$ and $j$ and one of several methods will be chosen to handle this. An example of one of these collision handling methods is described below. Once all this has been done control is handed back over to the Step method and the Update method is called. This Update method takes all of the temporary direction and position data that has been worked out by the collision check methods, and updated the actual positions and directions of the agents so that when the Move class updates the ring and the agents then they will appear to move around the ring governed by the rules set out by the model talked about earlier in this document.

for $i < \text{GUI.theta.length}$ do
   Put current positions of each agent on ring into a temporary array
end for
for $i < \text{GUI.direction.length}$ do
   Put current directions of each agent into temporary array
end for
for $i < \text{GUI.theta.length}$ do
   if GUI.direction[i] == 1 then
      move_Di1(i)
   end if
   if GUI.direction[i] == 0 then
      move_Di0(i)
   end if
end for
update()
3.4.4 move_Di1j0_Si1j1 - Agents class

This is an example of one of the collision checking methods that were mentioned above. Its function is to decide if a collision is going to occur in the next step if the agent continues to move in its current direction and at its current speed. Therefore the collision checking method that is called depends on an agent’s speed and current direction as well as the speed and current direction of the agent that it is currently being checked against. The example below shows the code that is run when agent $i$ is going clockwise at a speed of 1 and agent $j$ is going anti-clockwise at a speed of 1. The methods $\text{thetachkm}1$ and $\text{thetachkp}1$ are used to find out where the two agents will be on the ring if they move either clockwise, plus 1, or anti-clockwise, minus 1. Using this information the method can compare the two and decide if a collision will take place or not and also what agent $i$’s next position should be based on the rules of the model defined by this project.

The movement of $i$ in this example is given by calling $s1\text{ClockwiseMovement}(i)$, this method calculates the movement of the agent and the movement method that is called is based on what has been decided in this collision check method. After these temporary positions and directions have been updated control is given back to the Step method. Temporary directions and positions are used so that when the method loops back around and uses a different agent $a_i$ then all of the comparisons made per step are with the same data and this data is only updated after every agent has had its next move determined by the Agents class.

```java
if (thetachkp1(i) == GUI.theta[j]) then
    GUI.temptheta[i] = GUI.theta[i]
    GUI.tempdir[i] = 0
    colfound = TRUE
else
    if (thetachkp1(i)) == (thetachkm1(j)) then
        s1ClockwiseMovement(i)
        colound = TRUE
        GUI.tempdir[i] = 0
    else
        s1ClockwiseMovement(i)
    end if
end if
```

4 Design - Algorithm

4.1 Overview

The location discovery algorithm is formed of a number of stages. Each stage is a sequence of at most $n$ consecutive rounds. In the beginning of the first round of each stage an agent $a_i$ randomly chooses the direction (clockwise or anti-clockwise) of its movement. Later throughout the same stage the exact location and the movement direction of $a_i$ depends solely on the collisions with its neighbours $a_{i-1}$ and $a_{i+1}$. We show that on the conclusion of each round the agents always reside at the initial positions $p_0, ..., p_{n-1}$, where the current location of agent $a_i$ corresponds to $p_{i+k}$, for some $k \in 0, ..., n-1$. Note that this observation allows agents to visit (and record) the initial positions of other agents. A stage concludes when each agent $a_i$ arrives at the end of some round at its original position $p_i$. We show that with very high probability agents require $O(\log^2 n)$ stages to learn the locations of their counterparts. Since each stage is formed of at most $n$ rounds the total complexity of our algorithm is bounded by $O(n \log^2 n)$. When designing the discovery algorithm for this project we used the information obtained from the lemmas described earlier in this document. Due to Lemma 4 - Even $n$ and Lemma 5 - Odd $n$ we believed that by designing an algorithm that repeated this procedure of visiting starting locations until an agent arrives back at its original starting location when time $t$ is an integer value and then randomly choosing a new starting direction then the system will eventually be in the state described by Lemma 3 - Relative Primeness and therefore produce a situation where an agent
knows all of the relative starting locations of the other agents on the ring.

4.2 Pseudo-Code - Discovery Algorithm

The Pseudo-Code below is written from the point of view of an Agent on the ring that is defined in the Model. For the convenience of this description and the code itself the following variable names will be used:

\[ S = \text{Number of Stages required for solution} \]
\[ R = \text{Number of Rounds per stage} \]
\[ R_n = \text{Current Round} \]
\[ S_n = \text{Current Stage} \]
\[ T_d = \text{Traversal Direction \{Clockwise or Anti-Clockwise\}} \]
\[ D = \text{Relative distance from original starting position \{+ or -\}} \]
\[ N = \text{Number of Agents on Ring} \]
\[ S_t = \text{Steps taken by the agent during that round} \]
\[ T_u = \text{Time Unit \{One time unit is the number of steps it would take the agent to traverse an empty ring at its maximal speed\}} \]

The first part of the algorithm calls the Initialise sub-routine and passes it \( N \). In turn this sub-routine returns the values of both \( S \) and \( R \). The agent can then run the different loops in its code for the required number of times to obtain a solution. It is understood by all agents what is mean by Stage and Round i.e. a given number of Rounds make up one Stage and completing a stage means that the Agent has returned to its original starting position and it is also the end of the last round. At the beginning of each Stage \( D \) is reset to 0 as this will be the relative distance from the agents original starting position at this point. The current Round number will also be set to 0 at this point as a new Stage is beginning. Finally the agent will decide upon an initial starting direction by calling the sub-routine \textit{CoinFlip}. The agents share a common sense of what is meant by "Clockwise" and "Anti-Clockwise" and this sub-routine will randomly return one of these two directions with a probability of \( \frac{1}{2} \). Following on from this the agent then enters a block of code it will repeat until \( R_n = R \). This is because this code is what will be carried out during one Round and once the required number of rounds per Stage has been met then the code will break and a new Stage will begin. Within this block of code there is a loop that will be carried out for every step that the agent takes. Each step taken the agent will receive and analyse information provided by the system by calling the sub-routine \textit{CollectTrajectoryInformation}. It will then check if it has collided with another agent on the ring and if it has it will set its direction to the opposite of what it is at this point during the round. If a collision did happen then the agent will also carry out some computation about the position of the collision by calling the \textit{Compute} sub-routine. After this has been done the agent will then update its relative distance counter \( D \) and will do this by checking its \( T_d \) and then deciding if it should increment or decrement \( D \). Using this information the agent can again compute some information by calling the \textit{Compute} sub-routine. At this moment \( R_n \) will also be incremented. Once the agent has broken out of this loop it will then increment \( S_n \) until \( S_n \) is greater than \( S \). At this point S will have been met and with high probability a solution should have been found.

\[
\text{Initialise}(N) \\
\text{for } S_n < S \text{ do} \\
\quad D = 0 \\
\quad R_n = 0 \\
\quad T_d = \text{CoinFlip()} \\
\text{repeat} \\
\quad \text{for } S_t < T_u \text{ do} \\
\quad \quad \text{CollectTrajectoryInformation()} \\
\quad \text{end repeat} \\
\quad S_n = S_n + 1 \\
\text{end for}
\]
if $CollisionDetected()$ then
    if $T_d = "Clockwise"$ then
        $T_d = "Anti\,\!\,-\,\!\,Clockwise"$
    else
        $T_d = "Clockwise"$
    end if
    Compute()
end if

$S_t \leftarrow S_t + 1$

if $T_d = "Clockwise"$ then
    $D \leftarrow D + 1$
else
    $D \leftarrow D - 1$
end if

end for

Compute()

$R_n \leftarrow R_n + 1$

until $R_n = R$

$S_n \leftarrow S_n + 1$

end for

5 Realisation

5.1 Software

5.1.1 Overview

The software that has been produced was written in the Java programming language and allows the user to specify how many agents should be present in each simulation as well as view the steps of all the agents around the ring in real time. Other features that were included are the ability to set up a simulation where some agents travel at a different speed, there is also a feature to enable the user to view the state of the ring after each round has occurred and also a feature that keeps track in vector form of the positions of each agent at the start of each round.

![Home Screen Realisation](image)

Figure 9: Home Screen Realisation

Taking the feedback given from previous stages of the MSc project the software opens up to full screen
and the size of the text shown in the software has also increased from the prototype that had been produced previously.

### 5.1.2 Positions on the Ring

![Figure 10: Main Screen Realisation](image)

The positions on the ring that each agent can occupy were constructed using the formula for the equation of a circle, where the centre of the ring is at point $(0, 0)$ and points that the agents can occupy are calculated as $x = radius \times \cos t$ and $y = radius \times \sin t$. Therefore, this allows movement around the ring to be governed by the value of $t$ as the centre of the circle is at point $(0, 0)$.

### 5.1.3 Analysis Functions

There is an option to show the locations of each agent at the start of each round. This was included as a way to help prove some of our theories mentioned in the lemmas described previously. For example it shows that at the end of each round if the $gcd(i, n) = 1$ then each agent occupies a space that was occupied by an agent at the beginning of the simulation and that given enough rounds then this agent has in fact visited the starting locations of each agent on the ring at the end of each round. This is shown in the example below.

![Figure 11: Positional Data Menu Realisation](image)

To enable the above feature to work well with the human brain it was thought that having an image stored from the start of each round would provide a visual context to the data collected. This has been
achieved through the use of a screen shot feature in Java that allows items located within a certain JPanel to be captured in an image. These images are then available for viewing.

![Figure 12: Screenshot Menu Realisation](image)

5.1.4 Problems Occurred

The main problem that occurred while constructing this software was the logistics of writing the rules that governed the movement of the agents on the ring. This was not a problem for agents with a speed of 1 place per step along the ring. Where a step is the movement of the value of the agents speed along the ring and as the ring is split into 360 different points then one round takes 360 steps for an agent with speed equal to 1 to complete. The problem occurred when having multiple speeds of agents operating on the ring. The reason for this it turned out was that by having different speeds then collisions would occasionally happen on points that were between integer points on the ring. This presented a problem when writing the rules to govern this as these collision points were not fixed and every time they occurred then the next time they would become smaller and smaller fractions of space between the integer points. To solve this problem it was thought that by nominating a set of starting locations on the ring that were evenly distributed but would still be randomly populated by the agents then these mid-point collisions would not occur. It turned out that, for speeds of 1 and 2 then by dividing up the 360 points on the ring into segments of length 12 and having the start of these segments as the possible starting positions, then collisions would only occur either on integer points of exactly half way between these points. Due to this problem the only speed that the user can set the agents to be is either 1 or 2. However, this still provides the user with the ability to simulate what happens when two speeds are present in the model and it allows us to show that for the case where two different speeds are present on the ring our discovery algorithm will not function correctly.

5.2 Algorithm

The discovery algorithm designed for this project relied upon the lemmas described earlier holding. The software that has been produced alongside this project allows us to be able to see that these lemmas hold in reality as well as in theory. The following sequence of screenshots below show use of the software with 3 agents on the ring and a Rotational Index of 1. This sequence shows that Lemmas 1, 2 and 3 do in fact hold true.
The above screenshot shows the positional data of each agent at the start of each round. Notice that as the rounds progress each agent inherits the position of the agent that is one place anti-clockwise on the ring, shown here as the agent to its right.

The above screenshot shows the configuration of the agents on the ring at the start of the process.
The above screenshot shows the configuration of the agents after the first round.

The above screenshot shows the configuration of the agents after the second round.
The above screenshot shows the configuration of the agents after the third round.

It can be seen clearly that after each integer of time the agents have shifted by this Rotational Shift of $-1$ that is given by two agents moving anti-clockwise and one agent moving clockwise from their starting positions. As these lemmas seem to hold it can be assumed that the algorithm that has been designed based upon these lemmas will also hold and therefore yield a solution.

6 Evaluation

6.1 Testing

The testing that was carried out for this project was focused on using the software to back up the lemmas that have been outlined and proposed by this project. Testing for this project also meant running the software multiple times to see if any unusual circumstances occurred that may be interesting. This section of the dissertation focuses on the results of this experimentation with the software.

6.1.1 Rotational Shift

Sometimes when running a scenario where the number of agents on the ring was even there would be a case where the Rotational Shift produced was 0. When this occurred no matter how many rounds the software was left to run for the agents would always return to their original starting positions and so never have a chance to learn the starting locations of the other agents on the ring. It is for this reason that the Location Discovery algorithm outlined in this project uses a randomised procedure to decide upon a new traversal direction for the agent as this way we can ensure a high probability of eventually ending up in a situation where the Rotational Shift is -1 or 1 and so all of the starting locations can be visited.

![Figure 18: Positional Data Test 2](image)

The above screenshot shows the positional data of each agent at the start of each round when there are 4 agents on the ring and the Rotational Shift is 0. Notice that at the end of each round each agent returns to its original starting location.

6.1.2 Odd n

This randomised procedure of choosing a new direction, however, will only produce a state where every starting location is visited when $n$ is odd and so in the case when $n$ is even each agent must do some calculation as to how far away each agent is from the location it starts the round at. This can be done by keeping a record of the shortest distance travelled before a collision from the start of each round. In this way
it is possible for the agents to figure out, by simply doubling this distance, the location that the colliding agent started that round in. Therefore, in this way all relative starting locations will eventually be found. This is because the $gcd(i, n)$ will never be smaller than 2. As this is the case when $n$ is even we are aiming for a rotational shift of $-2$ or $2$, this is because when this case occurs the agent will visit every other starting location on the ring and will have a chance to calculate the minimum collision distance from each starting point during that Stage.

The above screenshot shows the configuration of the agents after the start of the process when $n = 4$ and the Rotational Shift is $-2$.

The above screenshot shows the configuration of the agents after the first round.
Figure 21: Screenshot Data Test 3

The above screenshot shows the configuration of the agents after the second round. Notice that after only 2 rounds the configuration of the agents is as it was at the start of the process. Meaning that each agent has missed half of the starting locations on the ring. This is because $n$ is even and the Rotational Shift is $-2$.

Figure 22: Screenshot Data Test 3

The above screenshot shows the configuration of the agents after the third round.
The above screenshot shows the configuration of the agents after the fourth round. Notice how after 4 rounds the configuration of the agents is as expected, back as it was at the start of the process.

6.1.3 Variable Speeds

The software that has been produced also enables the user to see data for when the agents are traveling on the ring at different speeds from each other. Of course as mentioned earlier it became apparent that it was a much greater challenge that what we had time for to incorporate many different speeds and so for the purpose of this version of the software it was decided that only the speeds of 1 and 2 would be implemented. This, however, still provides us with some interesting results.

The above screenshot shows the configuration of the agents after the start of the process when $n = 5$ and the Rotational Shift is $-1$. The speeds of the agents are not the same, two of the agents are running at a speed of 2 and the other three at a speed of 1.
The above screenshot shows the configuration of the agents at the end of the first round. Notice how this distribution is completely different from that of the initial distribution and because of this it is impossible to see if any shift has been made or not. This is the case for the subsequent rounds in this case.

The above screenshot shows the configuration of the agents at the end of the second round.
The above screenshot shows the configuration of the agents at the end of the third round.

The above screenshot shows the configuration of the agents at the end of the fourth round.
The above screenshot shows the configuration of the agents at the end of the fifth round.

![Screenshot Data Test 5](image)

Figure 30: Screenshot Data Test 5

The above screenshot shows the configuration of the agents at the end of the sixth round. If the case was that the speed of the agents were all the same, either all 1 or all 2 then this would be the round that all of the agents in this scenario would have returned to their original starting positions. However, this is not the case when the agents are traveling at different speeds. It has not been tested but it may be worthwhile running this case for a very large number of rounds to see if at any point the agents do return to their original starting locations at the end of a round.

![Positional Data Test 5](image)

Figure 31: Positional Data Test 5

The above screenshot shows the positional data collected from the above experiment. Notice how, although the agents do not take part in a shift the same way they do when their speeds are the same, the agents do occasionally visit locations at the end of rounds that other agents have occupied at the end of previous rounds. For example, in the screenshot above the starting position of Agent 1 in round 0, 181, is visited in round 5 by Agent 3. This is seen also when comparing the position of Agent 3 in round 2 and the position of Agent 4 in round 6, both 294. This is something that needs further investigation as it could lead to a possible way to obtain location discovery when the speeds of agents are not the same. This phenomenon has been observed several times throughout testing and the above case is only one example of this frequent occurrence when dealing with multiple speeds of the agents.
6.2 Strengths

The main strengths of this project can be split up into two main areas. They are as follows. Firstly, from a software standpoint, it has provided us with a way to visualise and model throughout the project and as such has given us a better view and understanding of the problem itself. Furthermore, we have been able to put any observations we have made to the test. This has allowed us to proceed with confidence when constructing an algorithm for location discovery. As an added extra the software that has been produced also has an option to be able to investigate and observe cases where the speeds of the agents on the ring are not the same. Although the software currently only supports the case when speeds of 1 and 2 are used this is a feature that will become useful for research that is beyond the scope of this project. A point of interest that occurred when running experiments on these types of cases was when two faster agents sandwiched a slower agent and all three made contact at the same point in time. The software enabled us to see what could happen based on the current model and therefore brought this problem to the forefront of this path of research as this is a scenario that will have to be investigated further and dealt with if this is a path we choose to explore in the future. From the research point of view the work that has been done allows us to continue along this path with confidence that a solution exists. The work that has been done up to this point is particularly interesting as it shows that it is in fact possible for a solution to be found, with regards to location discovery on the ring, without the agents having contact and having a very limited amount of starting knowledge. While carrying out this research it has become clear that a vast amount of open problems exist around this area and that in the future related work will have a large scope to look at. Such problems that have already been discussed briefly include the case when the agents initially do not know \( n \) and also the case when the agents are traveling at different speeds.

6.3 Weaknesses

From the main features from the specification of the program only one was not implemented. This is the acceleration feature that was to give the user the option to impose a rule on the movement of the agents such that after a collision occurred then the agent would have to gradually build up speed before it was able to travel at its maximum speed again. The reason for this omission was because of implementation difficulties associated with having multiple speeds of agents on the ring. For example, although the software allows the use of two different speeds, if upon collision then the agent would have to travel at half of its maximum speed for a certain number of steps before it achieved its maximum speed again then this then because sometimes an agent with a maximum speed of 1 would be traveling at a speed of 0.5 then it would not be possible to have collisions only occur at either whole integer or half points on the ring. Therefore gradually over time this fraction between two integer points that an agent will experience a collision on becomes very small and in this way creating a problem similar to that described above when working with different speeds of the agents.

7 Professional Issues

At least two page of discussion of how your project related to the British Computer Society Code of Conduct and Code of Good Practice.

7.1 Code of Conduct

There are four main areas that make up the British Computer Society Code of Conduct. These areas are mentioned below along with a short explanation as to how this project adhered to them.

7.1.1 Public Interest

This project does not interact with the public, nor does it store any personal information. Therefore all criteria in this section have been met.
7.1.2 Professional Competence and Integrity

The project was taken upon the understanding that I possessed the necessary skills to see it through to the end and complete work that was required. It should also be mentioned that throughout this project I have developed my skills, specifically with regards to improving my understanding of algorithmic techniques and obtaining a clearer understanding of Discrete Mathematic based solutions. Furthermore my skills in the Java programming language have also increased over the course of this project. Furthermore, throughout this work no plagiarism took place. All sources and references used can be found in the Bibliography section of this document. It should also be mentioned that this project came together through working with my supervisors, Prof. Leszek Gaśieniec and Dr. Russell Martin, and through input from Durham University's Dr Tom Friedetzky.

7.1.3 Duty to Relevant Authority

All laws enforced by the relevant authority have been adhered to throughout the course of this project and no attempts to misrepresent or withhold information or results have been made.

7.1.4 Duty to the Profession

Throughout this project the reputation of the profession has been upheld and no action which could bring the profession into disrepute has been taken. For example this work has been completed without plagiarism and all sources and references used have been cited and can be found in the Bibliography section of this document.

7.2 Code of Good Practice

This project complies with all standards in the British Computer Society's Code of Good Practice. Throughout the project I look to improve my relevant technical skills, these included skills based in the areas of Algorithms, Discrete Mathematics, GUI and Thread programming skills as well as learning and using the Latex and TeXworks environment throughout for all documentation produced. The workload and direction of this project was decided upon after initial meetings held with my supervisors. In this way the workload and area of work for each person reflected their strengths. A Gantt Chart time plan was used throughout the project to keep track of my progress and to plan the project properly by dividing it up into manageable parts that could be completed within specified widows of time. All ethical and legal concerns about this project are satisfied easily as there is no personal information held and the project itself does not directly interfere with the public or society. When undertaking this project it was ensured that quality management was employed. These standards where met throughout the project, in particular with the presentation of the simulation software as the GUI has been thoroughly tested so as to provide a high quality user experience. This document honestly summarises all mistakes encountered throughout the project and details what changes needed to be made to correct these errors. Also all results that have been reordered are also published with this document so that they are available for viewing. Finally, with regards to the research carried out in this project, the research carried out is, to our knowledge, only beneficial to society and the public as well as the university and in no way harms or discredits these parties. Also it is intended that the results of this research be shared with other researchers and organisations through the publication of a research paper on the topic covered by this project.

8 Conclusions

8.1 Project Outcome

This project set out with the aim to discover a way that agents on the ring, using the model described earlier, could in fact discover the starting locations of each other on the ring without communicating with
each other so that work could be done to provide a solution to the network patrolling problem using this model. This has been achieved in that the lemmas that were observed and throughout this project proven, something that had been aided by the software that has been produced for this project, have been used to provide an algorithm that would enable the agents to learn the relative starting locations of each other. This algorithm can be run for both the case when \( n \) is even and when \( n \) is odd. However, the minimum time complexity of the algorithm to provide a solution with high probability varies slightly depending upon what case it is. The minimum time complexity of a solution being found with very high probability when \( n \) is odd is \( O(n \log^2 n) \) and when \( n \) is even is also \( O(n \log^2 n) \). However, when \( n \) is even the best case that can be achieved is \( \gcd(i, n) = 2 \), meaning that only every other starting location gets visited at the end of each round. Therefore the other locations can only be discovered through calculating the smallest distance from the position the agent is starting in that round to when it next has a collision as the minimum distance when both agents that collide will be when they are traveling towards each other at the start of the round from their starting positions in that round and so, as the agents move at the same speed, by doubling the distance traveled since the start of that round an agent can determine where the other agent it is colliding with started during that round.

### 8.2 Further Work

Further expansion to the software may include being able to set the speed of an agent to a speed other than 1 or 2. It is believed that the same reasoning used to come up with the spacing of 12 between the possible starting locations for speeds of 1 and 2 can be applied to other speeds as well so as to include them in the simulation software in the future. Originally it was thought that given extra time then there would be some sort of distance time graph that could be computed and displayed, however it was thought that, given the nature of the problem and the theories we have made that, it would be best to show this movement in a vector format instead so as to get a more precise location on the ring for each agent. One feature that may be useful to implement is the colouring of the buttons in the screenshot menu to colour the one at the start of each stage the same colour so that it is clearer that the stage is complete and the agents should have arrived back at their original starting locations.

With regards to the discovery algorithm. We have some initial ideas of how to take this forward, some initial thoughts are on how to use the information learned in the discovery algorithm to get the agents to distribute themselves evenly on the ring and then patrol along the ring in a shared direction. There have been some discussions around how it may be possible for the agents to start off without knowing a common sense of direction. For this problem we consider separately cases with \( n = 1 \) and \( n = 2 \), and later we shall show how agents agree on the sense of direction for larger \( n \). We assume that when agents choose the movement direction they assume that this is the clockwise direction. For \( n = 1 \), trivially no agreement is required. For \( n \geq 2 \), however, the agreement mechanism is needed. Note that independently from the movement direction adopted by the two agents each stage lasts exactly one round, since in time one the agents always end up in their original positions. However, two types of stages (rounds) can be distinguished: (1) when the agents do not collide, and (2) when collisions occur. In order to agree on sense of direction a stage of type (1) must occur, i.e., when the agents do not collide they assume that the current direction is the clockwise direction. Since the movement direction adopted by each agent is chosen uniformly at random the probabilities associated with arriving in stages of type (1) and (2) are the same, and they are equal to \( 1/2 \). Thus after one stage, with the constant probability, and after \( l \) stages, with probability \( 1 - O(\frac{1}{n^r}) \), a stage of type (1) occurs and the agents agree on sense of direction. The case with \( n > 2 \) is more complex since one cannot await a stage of type (1), i.e., when the agents do not collide. The probability that all agents choose the same direction is \( \frac{2^n}{2^n} = \frac{1}{2^{n-r}} \), i.e., one would have to wait time exponential in \( n \) to agree on the sense of direction. In this case we focus on stages of type (2) with the rotation index \( r \neq 0 \). And when such stage occurs, the agents that see \( r > 0 \) do not change their understanding of the clockwise direction, and those with \( r < 0 \) replace the clockwise direction with its anticlockwise counterpart. The probability that a stage of type (2)
with \( r \neq 0 \) occurs is 1 when \( n \) is odd and \( 1 - \frac{(n/2)^2}{2^n} \gg 1/2 \) when \( n \) is even. Thus after \( l \) stages, i.e., after at most \( l \cdot n \) rounds the probability of having agreement on sense of direction is \( 1 - \frac{1}{2^l} \). Another interesting situation to investigate would be when the length of the ring itself is unknown by the agents. We know it is not possible to provide a solution to this using our current algorithm, however is there a way to do this at all or does a trade-off between either not knowing \( n \) or not knowing the length of the ring itself exist? We currently believe that it would be possible to learn the length of the ring if \( n \) is known this of course can be reversed and we believe that \( n \) can be learnt if the length of the ring is known and \( n \) is not as we believe that by counting the number of locations learnt through one round and also comparing these positions to the calculations made of distances from neighbours then an agent can figure out if all agents have been counted or not and therefore if \( n \) has been learnt. It is also thought that it is not possible to find a common sense of direction for the agents if the length of the ring is not known and \( n \) is also not known. This project will be leading into further study in the form of a PhD and so this area and related problems will be investigated over the coming years. Currently we are in the progress of writing a research paper on this topic that we are hoping to submit to a conference and the submission deadline is around the same time as the dissertation deadline.

References


10 Appendices

10.1 Appendix A - Original Specification Documentation
Specification

Project Title: Distributed Network Patrolling

Student:
Thomas Gorry

Supervisors:
Prof. Leszek Gąsieniec
Dr. Russell Martin

June 30, 2011
1 Project Description

1.1 The Problem

This project will be a research based project and its focus will be on that of Mobile Agent Coordination and Organisation in a network with reference to the Mobile Agent Patrolling problem. Specifically the project will look at the behaviour and a possible organisational solution of Agents situated on a ring. This area of research has its roots in the patrolling problem that has been studied by many in robotics [4] and has been looked into in the form of patrolling in a closed area by [7]. Network Patrolling is defined by [3] as a perpetual process performed in a static or in a dynamically changing environment. There has been different methods to measure the success of this in previous work. For example, in [7] they study patrolling in adversarial environments, in which the robots’ goal is to maximize their rewards. These rewards are received if the robots manage to observe the adversary, which tries to evade the patrolling robots and the more rewards the better the measurement of optimisation. For the purpose of this project algorithmic efficiency will be measured by its capacity to optimise the frequency of visits to each point of the ring, known as Idle Time. This is the general approach taken in this field [3,4]. For patrolling of a unidirectional circle with constant equal speed entities it is widely conjectured that the most efficient method, in terms of minimising Idle Time, is to arrange these patrolling entities at equal distances around the network and have them walk in a constant unified direction [3]. Therefore the Cyclic-Strategy of patrolling is the one that we have chosen for this project. This strategy is described by [2] and studied in [3]. There has been research in other directions, [5], that look towards the natural world to come up with a solution to the patrolling problem and have used the idea employed by ants of pheromone stamping and then having a "leader" Agent be responsible for finding the shortest path before employing a spreading algorithm for each Agent to separate to equal distances while following this pheromone trail. Unlike previous work in this area, that has included elements of communication between agents, swarm intelligence, distinct number of agents and so on, our research will assume a much simpler model described in more detail below and will look towards breaking the problem up into two phases. Firstly each agent locating the original starting position of every other agent and then entering a synchronisation phase before commencing the actual patrolling of the network. The synchronisation phase of this project is similar to that of the firing squad problem proposed by J. Myhill in 1957 [6] and it is likely that an answer lies in the study of this problem and its solutions.

1.2 The Model

Consider the following model of a ring with length 1 and n anonymous Agents situated randomly along its circumference. All Agents are allowed to walk along the ring in any direction (Clockwise, C, or Anti-Clockwise, A) where there is a universal notion of direction and time t. Initially this direction that the Agent begins to travel in is random. Each Agent has the abilities of measuring time, measuring distance, aware of n, and is collision aware. Furthermore each Agent has access to its own set of randomised bits that it can access if decisions need to be made. All Agents have the limitation of not being able to communicate with each other. We assume that while the set of n Agents are moving along the ring they have a maximal speed of 1 and that in $t = 1$ an Agent using this maximal speed makes exactly one complete cycle on the ring. Where Agents collide they change direction (instantaneously) and travel this new direction around the ring. Agents are able to move between 0 and their maximal speed and are able to choose to change direction or stop for some amount of time.

1.3 Aims and Objectives

The aim is to investigate and design protocols that minimize the time during which each point on the ring is visited. This will be achieved when the Agents have performed actions so they are arranged such
that traveling in a uniformed direction at their maximal speed then $\frac{1}{n}$ will be the idle time between each point on the ring being visited by an Agent. The project itself will be carried out as two parts concurrently run alongside each other. These two parts will be the theoretical section described above and a software element to the project will also be included. The software that will be developed alongside the theoretical portion of this project is intended to help with the modelling and visualisation of the problem as well as different proposed solutions in a simulator environment. The program will be designed and implemented based on the following requirements:

- The user should be able to visualise in real time what is happening on the ring and to each Agent.
- The user should be able to specify how many agents are present on the ring.
- The user should be able to have an option to specify a custom speed for each agent.
- The user should be able to have an option to employ acceleration on the Agents such that after collisions the Agents take some time to achieve maximum speed again.
- There should also be a function to show previous time periods in the animation.
- The program should be able to implement the final solution found by this project.

More comprehensive functions will be added to the software in the future in the event of the project scope shifting directions to include new spheres of interest. The software itself will be made available on the internet as a downloadable executable in the form of a .jar file so that those involved in the project will have easy access to the latest version. One of the reasons for the choice of a broad and easily adaptable set of options as part of the software is that this project is being used as a platform to PhD studies around this and similar problems. Therefore by having a wide-ranging program that gives the user the ability to choose different preferences to simulate then the project is able to be more inclusive of similar problems and different parameters, such as future investigation into the effect of acceleration after the collision of agents before reaching maximal speeds. Furthermore the option of customisable settings within the program will allow us to run experiments and analyse them to gain further intuition of the problem and possible solutions. There are also many practical applications for this research. Most research talks about the benefit that such algorithms would give to networks with Agents patrolling them and protecting them from adversaries [1,7]. Other areas that could benefit from this research would be that of network administrators with Agents patrolling their networks looking for faults and failures. Further still this could be applied to the field of surveillance with respect to equipment such as perimeter lights or cameras and even the field of garden lawn mowing and home cleaning robots.

1.4 Observations and Proposed Solution

While preparing for this project and during preliminary discussions there have been four main observations about a possible solution to this problem.

**Observation 1** Assuming that the Agents walk along the ring in a way described above then after $t = 1$, and ignoring any virtual names or labels given to the Agents, then the distribution is identical to that of the distribution of Agents at $t = 0$.

**Observation 2** Since the agents do not pass over each other then at all integers of $t$ each Agent reappears on the ring relative to their starting position with rotation of index $i = n_c - n_a$ where $n_c$ is the number of agents traversing the ring in a clockwise direction and $n_a$ is the number of Agents traversing the ring in an anti-clockwise direction.
Observation 3 Following on from the previous observation if \( \gcd(i, n) = 1 \), then in time \( t \) all agents return to their original location and they have also each learned the relative positions of all the other Agents in the set of Agents.

Observation 4 If \( n \) is odd then \( i \) is also odd and if \( n \) is even the index \( i \) is always even. This presents two problems. The first being if \( i \) is odd then the Agents must randomly decide to change direction and walk for a period of time before repeating the process in order to learn the relative starting locations of the other Agents on the ring. The second problem is when the index \( i \) is even. This presents a case where \( \gcd(i, n) \geq 2 \) so the above solution will only allow the Agent to learn half of the starting positions. The Agent can learn the other half of the starting locations must be learnt by keeping track of the minimum amount of time between each collision with its neighbours.

We believe that by repeating this procedure from each of the starting points learnt through Observation 3 will ensure with a high probability that each Agent will eventually know the relative starting points of all other Agents on the ring regardless of whether \( n \) is even or not. It is also believed that this can be achieved in a travelled distance of \( O(n \log^2 n) \)

2 Conduct of the Project

2.1 Required Software

For the purpose of this project a program will be developed using the Java Programming Language that will allow the modelling and visualisation of the problem. This program will be designed and implemented in the Eclipse Helios IDE on the Windows 7 Operating System. In addition the Sun Microsystems Java Development Kit (JDK) will be used for the compilation of the source code. A distribution of Latex will be used in conjunction with the TeXworks environment to create and edit all written documentation regarding this project. In order to organise this project effectively a time plan in the form of a Gantt Chart will be produced using Microsoft Project 2010. The text editor Notepad++ will be used in order to create a webpage that will contain downloadable content regarding this project so as to provide an up to date source of information and files for everyone involved in the project.

2.2 Required Skills and Learning Outcomes

At the end of this project the following skills will have been developed or had their understanding grown:

- Improvement and further understanding of algorithmic techniques.
- Develop a greater understanding of Discrete Mathematics, how they apply to Computer Science and how they can assist in problem solving and the implementation of solutions.
- Advancement of additional programming skills in the "Swing" library to develop a GUI for the project as well as a greater knowledge of "Thread" usage and management.
- Understanding of the installation and operation of Latex and TeXworks environments to produce required documentation for the project.

This project will also yield the required preparation for intended PhD studies regarding this problem in the field of Algorithms.
3 Statement of Deliverables

The main deliverables that should be present at the end of the project:

1. There will be a specification, design, and dissertation for this project.

2. An algorithmic solution to discover all relative starting positions of Agents on the ring in the model described earlier.

3. An algorithmic solution to use the information obtained by the previous deliverable for each Agent to position themselves equally around the ring and then all patrol in the same direction. This should be the most efficient way to minimise idle time between points on the ring and will most likely have its roots in the Firing Squad Problem [6].

4. Fully functional software that can model the environment, implement and display possible solutions as well as generally aid in the development of the first two deliverables.

5. Experiments run using the above mentioned software as well as analysis of results.

6. Also presented will be observations and possible proofs that were discussed in the preliminary meetings between supervisors and student.

4 Bibliography

References


10.2 Appendix B - Original Design Documentation and Slides
Design

Project Title: Distributed Network Patrolling

Student:
Thomas Gorry

Supervisors:
Prof. Leszek Gąsieniec
Dr. Russell Martin

July 25, 2011
1 Project Summary

1.1 The Problem

This project is a research based project and its focus is on that of Mobile Agent Coordination and Organisation in a network with reference to the Mobile Agent Patrolling problem. Specifically the project will look at the behaviour and a possible organisational solution of Agents situated on a ring. This area of research has its roots in the patrolling problem that has been studied by many in robotics [3] and has been looked into in the form of patrolling in a closed area by [6]. Network Patrolling is defined by [2] as a perpetual process performed in a static or in a dynamically changing environment. There have been different methods to measure the success of this in previous work. For example, in [6] they study patrolling in adversarial environments, in which the robots’ goal is to maximize their rewards. These rewards are received if the robots manage to observe the adversary, which tries to evade the patrolling robots and the more rewards the better the measurement of optimisation. For the purpose of this project algorithmic efficiency will be measured by its capacity to optimise the frequency of visits to each point of the ring, known as Idle Time. This is the general approach taken in this field [2, 3]. For patrolling of a unidirectional circle with constant equal speed entities it is widely conjectured that the most efficient method, in terms of minimising Idle Time, is to arrange these patrolling entities at equal distances around the network and have them walk in a constant unified direction [2]. Therefore the Cyclic-Strategy of patrolling is the one that we have chosen for this project. This strategy is described by [1] and studied in [2]. There has been research in other directions, [4], that look towards the natural world to come up with a solution to the patrolling problem and have used the idea employed by ants of pheromone stamping and then having a "leader" Agent be responsible for finding the shortest path before employing a spreading algorithm for each Agent to separate to equal distances while following this pheromone trail. Unlike previous work in this area, that has included elements of communication between agents, swarm intelligence, distinct number of agents and so on, our research will assume a much simpler model described in more detail below and will look towards breaking the problem up into two phases. Firstly each agent locating the original starting position of every other agent and then entering a synchronisation phase before commencing the actual patrolling of the network. The synchronisation phase of this project is similar to that of the firing squad problem proposed by J. Myhill in 1957 [5] and it is likely that an answer lies in the study of this problem and its solutions.

1.2 The Model

Consider the following model of a ring with length 1 and \( n \) anonymous Agents situated randomly along its circumference. All Agents are allowed to walk along the ring in any direction (Clockwise, C, or Anti-Clockwise, A) where there is a universal notion of direction and time \( t \). Initially this direction that the Agent begins to travel in is random. Each Agent has the abilities of measuring time, measuring distance, aware of \( n \), and is collision aware. Furthermore each Agent has access to its own set of randomised bits that it can access if decisions need to be made. All Agents have the limitation of not being able to communicate with each other. We assume that while the set of \( n \) Agents are moving along the ring they have a maximal speed of 1 and that in \( t = 1 \) an Agent using this maximal speed makes exactly one complete cycle on the ring. Where Agents collide they change direction (instantaneously) and travel this new direction around the ring. Agents are able to move between 0 and their maximal speed and are able to choose to change direction or stop for some amount of time.

2 Design - Simulation Software

2.1 Overview

The software that will be developed alongside the theoretical portion of this project is intended to help with the modelling and visualisation of the problem as well as different proposed solutions in a simulator.
environment. The program will have and eventually be tested against the following features:

- The user should be able to visualise in real time what is happening on the ring and to each Agent.
- The user should be able to specify how many agents are present on the ring.
- The user should be able to have an option to specify a custom speed for each agent.
- The user should be able to have an option to employ acceleration on the Agents such that after collisions the Agents take some time to achieve maximum speed again.
- There should also be a function to show previous time periods in the animation.
- The program should be able to implement the final solution found by this project.

More comprehensive functions will be added to the software in the future in the event of the project scope shifting directions to include new spheres of interest. Also it should be mentioned that due to the difficulty of simulating a continuous setting a discretized version of the ring will be used. This will mean that agents may only occupy integer positions along the circumference of the ring rather as opposed to the continuous ring being explored by the theoretical side of this project in which all positions on the ring being accessible.

2.2 Interface Design

Shown above is a screenshot of how the home screen will look when the software is finished. From here the user will be able to specify the number of agents to be included in the simulation as well as decide whether or not they wish to manually change the speeds of each agent or include acceleration of agents in the simulation. If the user chooses to include acceleration then when an agent collides with another it will gradually accelerate from a speed of 0 upon collision to its maximal speed. This feature, as well as that of including agents of different speeds will be incorporated with a view for further research into this area as this project is primarily concerned with agents of the same speed and in a model where acceleration does not take place.

Above is a screenshot of what the user will be asked if they choose the option of setting the speed of each agent.
Above is a screenshot of the main screen that the user will be shown when simulating the model. From here the user will be able to see an animation of what the ring environment looks like as the agents move along its circumference and change direction upon collision. It is hoped that this main screen will provide a way for possible patterns to be recognised and a visual testing feature for the solution to the Network Patrolling Problem described above. From this screen the users will be able to tell what Stage and Round the agents are currently in as well as turn the acceleration feature on or off. The user will also be able to choose to Start or Stop the system or start a new simulation, clicking this button will take the user back to the home screen where they will be able to set up a new scenario. There is also a button, and tabs at the top of the screen, that the user can click on to view the screenshot menu where still images of the ring when collisions occur are stored and can be viewed for analysis.

Shown above is a screenshot of the screenshot menu that the user will be shown when simulating the model. Clicking on any of the images on the right of the screen will show them in a large view in the main window. For each collision a still image of the ring will be taken and stored in this menu for further viewing and analysis. Therefore the thumbnail menu of screenshots will be dynamic in that it continuously changes size when a simulation is running; this is why it will be contained in a scrollable window. The user can use the tabs at the top of the main window to navigate back to the main animation screen.
2.3 UML Diagram

If required an online version of this class diagram can be found at http://www.csc.liv.ac.uk/m0tg/MSc.html
2.4 Pseudo-Code

For the purpose of this design document Pseudo-code and descriptions for interesting or particularly challenging methods will be included for the simulation software code. The methods that will be displayed here will include the paintComponent method from the GUI class; the run method from the Move class; and the run method from the Agents class.

paintComponent - GUI class
The paintComponent method is used to construct the ring that the agents will be traveling along as well as the agents themselves. In order for the system to work accurately the positions for each of the agents must also be accurate on the ring. Therefore for this reason the agents' positions are worked out using the Cosine rule and general Trigonometry. Working out the agents' positions on the ring in this way means that we can effectively label each point that can be visited on the rings circumference with the degree of angle the radius to that point would create between itself and the x-axis running through the centre point of the circle with the centre being the origin of (0,0). This way each point along the circumference of the ring can be represented by an integer between 0 and 359 (as 360 would be the same as position 0). The agents can then move along the ring in a similar fashion by simply changing their degree position by 1 each time, add 1 to go clockwise around the ring and minus 1 to go Anti-Clockwise. This refers to the discretized setting discussed earlier. However, this means that if the number of agents does not divide 360 then \( \frac{1}{n} \) placement of agents cannot be accomplished. Therefore some further thought must go into dealing with this issue.

\[
\begin{align*}
\text{ringColor} &= \text{white} \\
\text{getCenterCoordinate}_x() & \\
\text{getCenterCoordinate}_y() & \\
\text{getRadius() &}
\end{align*}
\[
\text{drawOval}((\text{centerCoordinate}_x, \text{centerCoordinate}_y, \text{radius} \times 2, \text{radius} \times 2))
\]
\[
\text{for } i < \text{numberOfAgents} \text{ do}
\]
\[
\begin{align*}
\text{positionOnTheRing} &= (\text{radius} \times (\cos(2 \times \pi \times (\text{theta}[i]/360)))) \\
\text{AgentColor} &= \text{Colors}[i] \\
i & \leftarrow i + 1
\end{align*}
\]
\[
\text{end for}
\]

run - Move class
When the user clicks on the "Start" button in the GUI then this method will be called as a new thread will be created to contain the creation of the desired number of agents and also the animation of the ring and the agents traversing along it. The loop in this code will keep running, and in doing so update the graphical output of the ring and the agents, until the user clicks the stop button and in turn the value of "terminate" becomes TRUE, this will then allow the thread to stop executing as it will break the While loop.

\[
\begin{align*}
\text{terminate} &= \text{FALSE} \\
\text{for } i < \text{numberofAgents} \text{ do}
\end{align*}
\]
\[
\begin{align*}
\text{Start a new thread that executes the Agents class} \\
i & \leftarrow i + 1
\end{align*}
\]
\[
\text{end for}
\]
\[
\text{loop}
\]
\[
\text{update ring display}
\]
\[
\text{if terminate = TRUE then}
\]
\[
\text{return}
\]
\[
\text{end if}
\]
run - Agents class
The run method in the Agents class will be called from the Move class each time a new agent is created. The agent will then loop around the code in this class executing the code that governs its movement on the ring. A similar mechanism has been used with this method as with the one above, when the user wants to stop the system they will click a button that will change the value of "agentRun" to FALSE and the thread will be allowed to break the loop and terminate. The method itself simply checks what direction that the agent is going in and its current position and then for all other agents on the ring it checks their direction and position against its own. With this knowledge the agent can decide if by continuing in its current direction then it will occupy the same point on the ring as another agent. If the agent thinks that this is the case then it immediately changes direction and if the agent it is going to collide with is going the opposite direction to begin with then it will also switch direction. Once all new directions have been determined then the method updates the agent's position by moving it by one position along the ring's circumference in its current direction. If there is a scenario where by if both agents move one position towards each other then they will occupy the same position then the system automates this by altering the animation to show that by moving one space the agents have actually moved 0.5 towards each other and 0.5 away from each other, giving the view that in the time it takes to move 1 space the agents are in the same position as they were previously. This is why there is a pause feature shown below in order to simulate this behaviour. Finally the method has a sleep statement at the end of the loop that will sleep for a time that is dependent on the speed of the agent (this can be left as a standard value or altered by the user from the GUI).

agentRun = FALSE
while agentRun = TRUE do
    position = get Agents current position on the ring from control class
    Direction = get Agents current traversal direction on the ring from control class
    if Direction = clockwise then
        for j < numberOfAgents do
            jDirection = Direction of agent j
            jPosition = Position of agent j
            if jDirection = clockwise then
                if (position + 1) = jposition then
                    Direction = Anti-Clockwise
                    Take a screen capture of ring as collision has occurred.
                end if
            else
                if (position+1) = jposition then
                    Direction = Anti-Clockwise
                    jDirection = Clockwise
                    Take a screen capture of ring as collision has occurred.
                else
                    if (position + 1) = (jposition - 1) then
                        Direction = Anti-Clockwise
                        jDirection = Clockwise
                        Take a screen capture of ring as collision has occurred.
                        Both agents pause for 1 move.
                    end if
                end if
            end if
        end for
    end if
    j ← j + 1
end while
else
if Direction = Anti-Clockwise then
for j < numberOfAgents do
jDirection = Direction of agent j
jPosition = Position of agent j
if jDirection = Anti-Clockwise then
if (position - 1) = jposition then
Direction = Clockwise
Take a screen capture of ring as collision has occurred.
end if
else
if (position-1) = jposition then
Direction = Clockwise
jDirection = Anti-Clockwise
Take a screen capture of ring as collision has occurred.
else
if (position - 1) = (jposition + 1) then
Direction = Clockwise
jDirection = Anti-Clockwise
Take a screen Capture of ring as collision has occurred.
Both agents pause for 1 move.
end if
end if
end if
end for
end if
end if
end if
update Direction in control class
if Pause = TRUE then
Pause = FALSE
return
else
if Pause = FALSE then
if Direction = Anti-Clockwise then
if Position > 0 then
Position ← Possition - 1
else
if Position = 0 then
Position = (360-1)
end if
end if
else
if Direction = Clockwise then
if Position < 360 then
Position ← Possition + 1
else
if Position = 360 then
Position = 1
end if
end if
end if
end if
end if
end if
end if
3 Design - Algorithm

3.1 Overview

The aim is to investigate and design protocols that minimize the time during which each point on the ring is visited. This will be achieved when the Agents have performed actions so they are arranged such that traveling in a uniformed direction at their maximal speed then \( \frac{1}{n} \) will be the idle time between each point on the ring being visited by an Agent. The project itself will be carried out as two parts concurrently run alongside each other. These two parts will be the simulation software element described above and a theoretical section.

Lemma 1 Assuming that the Agents walk along the ring in a way described above then after \( t = 1 \), and ignoring any virtual names or labels given to the Agents, then the distribution is identical to that of the distribution of Agents at \( t = 0 \).

Lemma 2 Since the agents do not pass over each other then at all integers of \( t \) each Agent reappears on the ring relative to their starting position with rotation of index \( i = n_c - n_a \) where \( n_c \) is the number of agents traversing the ring in a clockwise direction and \( n_a \) is the number of Agents traversing the ring in an anti-clockwise direction.

Lemma 3 Following on from the previous observation if \( \gcd(i, n) = 1 \), then in time \( t \) all agents return to their original location and they have also each learned the relative positions of all the other Agents in the set of Agents.

Lemma 4 If \( n \) is odd then \( i \) is also odd and if \( n \) is even the index \( i \) is always even. This presents two problems. The first being if \( i \) is odd then the Agents must randomly decide to change direction and walk for a period of time before repeating the process in order to learn the relative starting locations of the other Agents on the ring. The second problem is when the index \( i \) is even. This presents a case where \( \gcd(i, n) \geq 2 \) so the above solution will only allow the Agent to learn half of the starting positions. The Agent can learn the other half of the starting locations must be learnt by keeping track of the minimum amount of time between each collision with its neighbours.

We believe that by repeating this procedure from each of the starting points learnt through Lemma 3 will ensure with a high probability that each Agent will eventually know the relative starting points of all other Agents on the ring regardless of whether \( n \) is even or not. It is also believed that this can be achieved in a traversed distance of \( O(n\log^2{n}) \)
3.2 Pseudo-Code - Discovery Algorithm

The Pseudo-Code below is written from the point of view of an Agent on the ring that is defined in the Model. For the convenience of this description and the code itself the following variable names will be used:

- \( S \) = Number of Stages required for solution
- \( R \) = Number of Rounds per stage
- \( R_n \) = Current Round
- \( S_n \) = Current Stage
- \( T_d \) = Traversal Direction \{Clockwise or Anti-Clockwise\}
- \( D \) = Relative distance from original starting position \{+ or -\}
- \( N \) = Number of Agents on Ring
- \( S_t \) = Steps taken by the agent during that round
- \( T_u \) = Time Unit \{One time unit is the number of steps it would take the agent to traverse an empty ring at its maximal speed\}

The first part of the algorithm calls the Initialise sub-routine and passes it \( N \). In turn this sub-routine returns the values of both \( S \) and \( R \). The agent can then run the different loops in its code for the required number of times to obtain a solution. It is understood by all agents what is mean by Stage and Round i.e. a given number of Rounds make up one Stage and completing a stage means that the Agent has returned to its original starting position and it is also the end of the last round. At the beginning of each Stage \( D \) is reset to 0 as this will be the relative distance from the agents original starting position at this point. The current Round number will also be set to 0 at this point as a new Stage is beginning. Finally the agent will decide upon an initial starting direction by calling the sub-routine CoinFlip. There is a common understanding between all agents of what is mean by "Clockwise" and "Anti-Clockwise" and this sub-routine will randomly return one of these two directions with a probability of \( \frac{1}{2} \). Following on from this the agent then enters a block of code it will repeat until \( R_n = R \). This is because this code is what will be carried out during one Round and once the required number of rounds per Stage has been met then the code will break and a new Stage will begin. Within this block of code there is a loop that will be carried out for every step that the agent takes. Each step taken the agent will receive and analyse information provided by the system by calling the sub-routine CollectTrajectoryInformation. It will then check if it has collided with another agent on the ring and if it has it will set its direction to the opposite of what it is at this point during the round. If a collision did happen then the agent will also carry out some computation about the position of the collision by calling the Compute sub-routine. After this has been done the agent will then update its relative distance counter \( D \) and will do this by checking its \( T_d \) and then deciding if it should increment or decrement \( D \). Using this information the agent can again compute some information by calling the Compute sub-routine. At this moment \( R_n \) will also be incremented. Once the agent has broken out of this loop it will then increment \( S_n \) until \( S_n \) is greater than \( S \). At this point \( S \) will have been met and with high probability a solution should have been found.

Initialise(N)

\[
\text{for } S_n < S \text{ do}
\]

\[
D = 0
\]

\[
R_n = 0
\]

\[
T_d = \text{CoinFlip()}
\]

\[
\text{repeat}
\]

\[
\text{for } S_t < T_u \text{ do}
\]

\[
\text{CollectTrajectoryInformation()}
\]

\[
\text{if } \text{CollisionDetected()} \text{ then}
\]

\[
\text{if } T_d = \text{"Clockwise" then}
\]
\[ T_d = \text{"Anti - Clockwise"} \]  
else  
\[ T_d = \text{"Clockwise"} \]  
end if  
Compute()  
end if  
\[ S_t \leftarrow S_t + 1 \]  
if \[ T_d = \text{"Clockwise"} \] then  
\[ D \leftarrow D + 1 \]  
else  
\[ D \leftarrow D - 1 \]  
end if  
end for  
Compute()  
\[ R_n \leftarrow R_n + 1 \]  
until \[ R_n = R \]  
\[ S_n \leftarrow S_n + 1 \]  
end for

## 4 Testing Plan

These are tests that shall be run on the simulation software:

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Termination Check</td>
<td>Does the simulation software terminate correctly?</td>
</tr>
<tr>
<td>Environment Check</td>
<td>Does the simulation software create a ring and the environment described by the model?</td>
</tr>
<tr>
<td>Number of Agents check</td>
<td>Does the simulation software allow for user input for the number of agents on the ring?</td>
</tr>
<tr>
<td>Speed Check</td>
<td>Does the simulation software allow the user to dictate the speed of each agent?</td>
</tr>
<tr>
<td>Acceleration Check</td>
<td>Is there an option for the user to specify the use of acceleration in the simulation?</td>
</tr>
<tr>
<td>Analysis Check</td>
<td>Does the simulation software allow for the user to view screenshots of each collision so as to be able to analyse these occurrences?</td>
</tr>
<tr>
<td>Robustness 1 Check</td>
<td>Can the simulation software be used on other terminals other than the one it was designed on?</td>
</tr>
<tr>
<td>Robustness 2 Check</td>
<td>Is the simulation software reusable and can it be paused, stopped and started at any time during a simulation?</td>
</tr>
<tr>
<td>Robustness 3 Check</td>
<td>Does the simulation software allow the user to implement the discovery algorithm described above?</td>
</tr>
<tr>
<td>Robustness 4 Check</td>
<td>Can the simulation software run a solution to the Patrolling Problem described earlier?</td>
</tr>
</tbody>
</table>

These are tests that shall be performed on the solution to the Network Patrolling Problem that was laid out earlier:


<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Check 1</td>
<td>Does the algorithm enable all the agents on the ring to find the relative</td>
</tr>
<tr>
<td></td>
<td>starting positions of the other agents if there are an odd number of agents</td>
</tr>
<tr>
<td></td>
<td>on the ring?</td>
</tr>
<tr>
<td>Function Check 2</td>
<td>Does the algorithm enable all the agents on the ring to find the relative</td>
</tr>
<tr>
<td></td>
<td>starting positions of the other agents if there is an even number of agents</td>
</tr>
<tr>
<td></td>
<td>on the ring?</td>
</tr>
<tr>
<td>Speed Check</td>
<td>Does the solution perform this task with high probability in a time of</td>
</tr>
<tr>
<td></td>
<td>$O(n\log^2 n)$</td>
</tr>
<tr>
<td>Acceleration Check</td>
<td>Does the solution still work when acceleration of agents is taken into</td>
</tr>
<tr>
<td></td>
<td>account?</td>
</tr>
<tr>
<td>Robustness Check 1</td>
<td>Does the solution work for both small and large values of $n$?</td>
</tr>
</tbody>
</table>

Further testing may be carried out and will be recorded in the final dissertation. In terms of analysis of the movement of the agents when the Discovery algorithm is run on them the user can use the Simulation Software to track each agents collisions by looking at the screenshot menu provided by the software. If there is time further analysis tools will be included in the software. One that is currently being considered is a graph structure to monitor the agents’ movement based on the travel graphs visioned by E. J. Marey [7].

5 Review Against Plan

For this project no changes from the original project have been made and progress is as expected as laid out in the time plan presented in the specification. For a more detailed analysis of progress to date a full Gantt Chart with marked progress can be found at http://www.csc.liv.ac.uk/m0tg/MSc.html

6 Bibliography

References


MSc Project Design
Distributed Network Patrolling

Thomas Gorry  Prof. Leszek Gąsieniec
Dr Russell Martin

Model
• Synchronised model of a ring with anonymous agents randomly spaced along the circumference.
• No exchange of messages.
• Maximal speed of 1

Task
• Trajectory discovery of all participating agents
• Interesting variant of building a static map of the discovered environment.

Pseudo-code
procedure Discovery(S: length of the process);
    location := 0;
    for $R = 0$ to $S$ do
        if (location = 0) then $T_0 := \text{CoinFlip}();$ /* choose C or A direction */
        Location := (Location + shift := Traverse&Collect($T_0))$ modulo 1;
    end for;
end Discovery;
Time Complexity

- High probability that the Time Complexity is $O(n \log^2 n)$

- Relation to Coupon Collection Problem so at least $O(n \log n)$

- If $n$ is even then $\log n$ coin flips will give success with a high probability.
### Plan of Work

<table>
<thead>
<tr>
<th>Design</th>
<th>20/09/2011</th>
<th>26/02/2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>20/09/2011</td>
<td>26/02/2012</td>
</tr>
<tr>
<td>Analysis</td>
<td>20/09/2011</td>
<td>26/02/2012</td>
</tr>
<tr>
<td>Previews</td>
<td>20/09/2011</td>
<td>26/02/2012</td>
</tr>
<tr>
<td>Protection and Documentation Deadline</td>
<td>20/09/2011</td>
<td>26/02/2012</td>
</tr>
</tbody>
</table>

**Software Implementation and Testing**

| Software 1.1 | 20/09/2011 | 26/02/2012 |
| Software 1.2 | 20/09/2011 | 26/02/2012 |
| Software 1.3 | 20/09/2011 | 26/02/2012 |
| Software 1.4 | 20/09/2011 | 26/02/2012 |
| Software 1.5 | 20/09/2011 | 26/02/2012 |

**Documentation and Analysis of Results**

| Analysis | 20/09/2011 | 26/02/2012 |
| Test Runs | 20/09/2011 | 26/02/2012 |
| Report | 20/09/2011 | 26/02/2012 |
| Evaluation | 20/09/2011 | 26/02/2012 |
| Communication | 20/09/2011 | 26/02/2012 |

**Testing and Implementation**

| Testing and Implementation | 20/09/2011 | 26/02/2012 |
| Final Preparation | 20/09/2011 | 26/02/2012 |
| Final Submission | 20/09/2011 | 26/02/2012 |

---

**Thank You**
10.3 Appendix C - Original Final Presentation Documentation and Slides
Final Presentation Documentation

Project Title: Distributed Agents Location Discovery on the Ring

Student:
Thomas Gorry

Supervisors:
Prof. Leszek Gašieniec
Dr. Russell Martin

August 26, 2011
1 Project Summary

1.1 The Problem

This project is a research based project and its focus is on that of Mobile Agent Coordination and Organisation in a network with reference to the Mobile Agent Patrolling problem. Specifically the project will look at the behaviour and a possible organisational solution of Agents situated on a ring. This area of research has its roots in the patrolling problem that has been studied by many in robotics [3] and has been looked into in the form of patrolling in a closed area by [6]. Network Patrolling is defined by [2] as a perpetual process performed in a static or in a dynamically changing environment. There have been different methods to measure the success of this in previous work. For example, in [6] they study patrolling in adversarial environments, in which the robots' goal is to maximise their rewards. These rewards are received if the robots manage to observe the adversary, which tries to evade the patrolling robots and the more rewards the better the measurement of optimisation. For the purpose of this project algorithmic efficiency will be measured by its capacity to optimise the frequency of visits to each point of the ring, known as Idle Time. This is the general approach taken in this field [2,3]. For patrolling of a unidirectional circle with constant equal speed entities it is widely conjectured that the most efficient method, in terms of minimising Idle Time, is to arrange these patrolling entities at equal distances around the network and have them walk in a constant unified direction [2]. Therefore the Cyclic-Strategy of patrolling is the one that we have chosen for this project. This strategy is described by [1] and studied in [2]. There has been research in other directions, [4], that look towards the natural world to come up with a solution to the patrolling problem and have used the idea employed by ants of pheromone stamping and then having a "leader" Agent be responsible for finding the shortest path before employing a spreading algorithm for each Agent to separate to equal distances while following this pheromone trail. Unlike previous work in this area, that has included elements of communication between agents, swarm intelligence, distinct number of agents and so on, our research will assume a much simpler model described in more detail below and will look towards breaking the problem up into two phases. Firstly each agent locating the original starting position of every other agent and then entering a synchronisation phase before commencing the actual patrolling of the network. The synchronisation phase of this project is similar to that of the firing squad problem proposed by J. Myhill in 1957 [5] and it is likely that an answer lies in the study of this problem and its solutions.

1.2 The Model

Consider the following model of a ring with length 1 and \( n \) anonymous Agents situated randomly along its circumference. All Agents are allowed to walk along the ring in any direction (Clockwise, C, or Anti-Clockwise, A) where there is a universal notion of direction and time \( t \). Initially this direction that the Agent begins to travel in is random. Each Agent has the abilities of measuring time, measuring distance, aware of \( n \), and is collision aware. Furthermore each Agent has access to its own set of randomised bits that it can access if decisions need to be made. All Agents have the limitation of not being able to communicate with each other. We assume that while the set of \( n \) Agents are moving along the ring they have a maximal speed of 1 and that in \( t = 1 \) an Agent using this maximal speed makes exactly one complete cycle on the ring. Where Agents collide they change direction (instantaneously) and travel this new direction around the ring. Agents are able to move between 0 and their maximal speed and are able to choose to change direction or stop for some amount of time.

2 Design Summary

The design proposed two distinct areas that would be implemented. The first being software that allowed simulation of the model and the second being theoretical research of an algorithm to solve the problem summarised above.
2.1 Software

In the design it was said that the software would have the following specifications:

- The problem should be made visual in real time by the software using the model described earlier in this document.
- The number of agents on the ring should be variable.
- There should be an option to have agents walking at different speeds.
- There should be an option to have agents gradually accelerate up to their top speeds after collisions.
- There should be some visual of data from different periods of time that the user can analyse.

2.2 Algorithm

Based on the following lemmas, the design stated that an algorithm would be designed and researched that would enable agents to minimize the time during which each point on the ring is visited. This being achieved when the agents have performed actions so that they are arranged such that traveling in a uniformed direction at their maximal speed then \( \frac{n}{n} \) will be the idle time between each point on the ring being visited by an Agent.

**Lemma 1** Assuming that the Agents walk along the ring in a way described above then after \( t = 1 \), and ignoring any virtual names or labels given to the Agents, then the distribution is identical to that of the distribution of Agents at \( t = 0 \).

**Lemma 2** Since the agents do not pass over each other then at all integers of \( t \) each Agent reappears on the ring relative to their starting position with rotation of index \( i = n_c - n_a \) where \( n_c \) is the number of agents traversing the ring in a clockwise direction and \( n_a \) is the number of Agents traversing the ring in an anti-clockwise direction.

**Lemma 3** Following on from the previous observation if \( \gcd(i, n) = 1 \), then in time \( t \) all agents return to their original location and they have also each learned the relative positions of all the other Agents in the set of Agents.

**Lemma 4** If \( n \) is odd then \( i \) is also odd and if \( n \) is even then the index \( i \) is always even. This presents two problems. The first being if \( i \) is odd then the Agents must randomly decide to change direction and walk for a period of time before repeating the process in order to learn the relative starting locations of the other Agents on the ring. The second problem is when the index \( i \) is even. This presents a case where \( \gcd(i, n) \geq 2 \) so the above solution will only allow the Agent to learn half of the starting positions. The Agent can learn the other half of the starting locations must be learnt by keeping track of the minimum amount of time between each collision with its neighbours.

It was also stated in the design that we believed that this procedure from each of the starting points learnt through **Lemma 3** would ensure with a high probability that each Agent will eventually know the relative starting points of all other Agents on the ring regardless of whether \( n \) is even or not and that this could be achieved in a traversed distance of \( O(n \log^2 n) \)

3 Work Implemented and Changes to Design

3.1 Discovery Algorithm

The algorithm shown below, from the point of view of an agent on the ring, has not been changed. However, the time complexity of the algorithm has been altered following investigation.

For the convenience of this description and the code itself the following variable names will be used:
\[ S = \text{Number of Stages required for solution} \]
\[ R = \text{Number of Rounds per stage} \]
\[ R_n = \text{Current Round} \]
\[ S_n = \text{Current Stage} \]
\[ T_d = \text{Traversal Direction \{Clockwise or Anti-Clockwise\}} \]
\[ D = \text{Relative distance from original starting position \{+ or -\}} \]
\[ N = \text{Number of Agents on Ring} \]
\[ S_t = \text{Steps taken by the agent during that round} \]
\[ T_u = \text{Time Unit \{One time unit is the number of steps it would take the agent to traverse an empty ring at its maximal speed\}} \]

The first part of the algorithm calls the Initialise sub-routine and passes it \( N \). In turn this sub-routine returns the values of both \( S \) and \( R \). The agent can then run the different loops in its code for the required number of times to obtain a solution. It is understood by all agents what is meant by Stage and Round i.e. a given number of Rounds make up one Stage and completing a stage means that the Agent has returned to its original starting position and it is also the end of the last round. At the beginning of each Stage \( D \) is reset to 0 as this will be the relative distance from the agents original starting position at this point. The current Round number will also be set to 0 at this point as a new Stage is beginning. Finally the agent will decide upon an initial starting direction by calling the sub-routine \textit{CoinFlip}. There is a common understanding between all agents of what is meant by "Clockwise" and "Anti-Clockwise" and this sub-routine will randomly return one of these two directions with a probability of \( \frac{1}{2} \). Following on from this the agent then enters a block of code it will repeat until \( R_n = R \). This is because this code is what will be carried out during one Round and once the required number of rounds per Stage has been met then the code will break and a new Stage will begin. Within this block of code there is a loop that will be carried out for every step that the agent takes. Each step taken the agent will receive and analyse information provided by the system by calling the sub-routine \textit{CollectTrajectoryInformation}. It will then check if it has collided with another agent on the ring and if it has it will set its direction to the opposite of what it is at this point during the round. If a collision did happen then the agent will also carry out some computation about the position of the collision by calling the \textit{Compute} sub-routine. After this has been done the agent will then update its relative distance counter \( D \) and will do this by checking its \( T_d \) and then deciding if it should increment or decrement \( D \). Using this information the agent can again compute some information by calling the \textit{Compute} sub-routine. At this moment \( R_n \) will also be incremented. Once the agent has broken out of this loop it will then increment \( S_n \) until \( S_n \) is greater than \( S \). At this point S will have been met and with high probability a solution should have been found.

Initialise(N)
for \( S_n < S \) do
  \[ D = 0 \]
  \[ R_n = 0 \]
  \[ T_d = \textit{CoinFlip()} \]
repeat
  for \( S_t < T_u \) do
    CollectTrajectoryInformation()
    if \textit{CollisionDetected()} then
      if \( T_d = \text{"Clockwise"} \) then
        \[ T_d = \text{"Anti – Clockwise"} \]
      else
        \[ T_d = \text{"Clockwise"} \]
    end if
    Compute()
end if
$S_t \leftarrow S_t + 1$
if $T_d = "Clockwise"$ then
  $D \leftarrow D + 1$
else
  $D \leftarrow D - 1$
end if
end for
Compute()
$R_n \leftarrow R_n + 1$
until $R_n = R$
$S_n \leftarrow S_n + 1$
end for

The original estimation for this randomized protocol to achieve success was $O(n \log^2 n)$. However, as success can be defined as when the $\gcd(k, n) = 1$, where $k = (n_c - n_a)$ and $n = \text{number of agents}$, then for the agents to achieve success, in the case when $n$ is odd, then the number of rounds that must take place is $O(n^{\frac{3}{2}} \log n)$ and the number of stages will be $O(\sqrt{n \log n})$. In cases where $n$ is even we believe that the time taken may include another logarithm and so this would bring the time complexity to $O(\sqrt{n \log^2 n})$.

3.1.1 Experiments

The software produced for this project provides a supporting role for the experiments that will be carried out. These experiments will be the analysis of the results provided by the software to help prove the theories laid out in the lemmas above. There will also be some experiments run on cases where agents have different speeds to see if these theories fit with this new scenario.

3.2 Simulation Software

3.2.1 Overview

The software that has been produced was written in the Java programming language and allows the user to specify how many agents should be present in each simulation as well as view the steps of all the agents around the ring in real time. Other features that were included are the ability to set up a simulation where some agents travel at a different speed, there is also a feature to enable the user to view the state of the ring after each round has occurred and also a feature that keeps track in vector form of the positions of each agent at the start of each round.
Taking the feedback given from previous stages of the MSc project the software opens up to full screen and the size of the text shown in the software has also increased from the prototype that had been produced previously.

### 3.2.2 Positions on the Ring

![Diagram of ring positions](image)

The positions on the ring that each agent can occupy were constructed using the formula for the equation of a circle, where the centre of the ring is at point $(0, 0)$ and points that the agents can occupy are calculated as $x = radius \times \cos t$ and $y = radius \times \sin t$. Therefore, this allows movement around the ring to be governed by the value of $t$ as the centre of the circle is at point $(0, 0)$.

### 3.2.3 Analysis Functions

There is an option to show the locations of each agent at the start of each round. This was included as a way to help prove some of our theories mentioned in the lemmas described previously. For example it shows that at the end of each round if the $gcd(i, n) = 1$ then each agent occupies a space that was occupied by an agent at the beginning of the simulation and that given enough rounds then this agent has in fact visited the starting locations of each agent on the ring at the end of each round. This is shown in the example below.

![Positions table](image)

To enable the above feature to work well with the human brain it was thought that having an image stored from the start of each round would provide a visual context to the data collected. This has been achieved
through the use of a screen shot feature in Java that allows items located within a certain JPanel to be captured in an image. These images are then available for viewing.

![Image](image.jpg)

3.2.4 Problems Occurred

The main problem that occurred while constructing this software was the logistics of writing the rules that governed the movement of the agents on the ring. This was not a problem for agents with a speed of 1 place per step along the ring. Where a step is the movement of the value of the agents speed along the ring and as the ring is split into 360 different points then one round takes 360 steps for an agent with speed equal to 1 to complete. The problem occurred when having multiple speeds of agents operating on the ring. The reason for this it turned out was that by having different speeds then collisions would occasionally happen on points that were between integer points on the ring. This presented a problem when writing the rules to govern this as these collision points were not fixed and every time they occurred then the next time they would become smaller and smaller fractions of space between the integer points. To solve this problem it was thought that by nominating a set of starting locations on the ring that were evenly distributed but would still be randomly populated by the agents then these mid-point collisions would not occur. It turned out that, for speeds of 1 and 2 then by dividing up the 360 points on the ring into segments of length 12 and having the start of these segments as the possible starting positions, then collisions would only occur either on integer points of exactly half way between these points. Due to this problem the only speed that the user can set the agents to be is either 1 or 2. However, this still provides the user with the ability to simulate what happens when two speeds are present in the model and it allows us to show that for the case where two different speeds are present on the ring our discovery algorithm will not function correctly.

4 Evaluation

4.1 Shortfalls

From the main features from the specification of the program only one was not implemented. This is the acceleration feature that was to give the user the option to impose a rule on the movement of the agents such that after a collision occurred then the agent would have to gradually build up speed before it was able to travel at its maximum speed again. The reason for this omission was because of implementation difficulties associated with having multiple speeds of agents on the ring. For example, although the software allows the use of two different speeds, if upon collision then the agent would have to travel at half of its maximum speed for a certain number of steps before it achieved its maximum speed again then this then because sometimes an agent with a maximum speed of 1 would be traveling at a speed of 0.5 then it would not be possible to have collisions only occur at either whole integer or half points on the ring. Therefore gradually over time
this fraction between two integer points that an agent will experience a collision on becomes very small and
in this way creating a problem similar to that described above when working with different speeds of the
agents.

4.2 Future Development

Further expansion to the software may include being able to set the speed of an agent to a speed other that
1 or 2. It is believed that the same reasoning used to come up with the spacing of 12 between the possible
starting locations for speeds of 1 and 2 can be applied to other speeds as well so as to include them in the
simulation software in the future. Originally it was thought that given extra time then there would be some
sort of distance time graph that could be computed and displayed, however it was thought that, given the
nature of the problem and the theories we have made that, it would be best to show this movement in a
vector format instead so as to get a more precise location on the ring for each agent. One feature that we
would like to have completed by the final presentation itself is the colouring of the buttons in the screenshot
menu to colour the one at the start of each stage the same colour so that it is clearer that the stage is
complete and the agents should have arrived back at their original starting locations.

With regards to the discovery algorithm. We have some initial ideas of how to take this forward, some
of which include that it may be possible for the agents to carry out a few more quick and simple tasks that
would eliminate the need for the agents to know $n$ or indeed even have a common sense of direction as it is
thought that by carrying out some simple procedures the agents can in fact learn this information as well,
all without any communication other than colliding with each other. We also have some initial thoughts
on how to use the information learned in the discovery algorithm to get the agents to distribute themselves
evenly on the ring and then patrol along the ring in a shared direction. This project will be leading into
further study in the form of a PhD and so this area are related problems will be investigated over then
coming years. Currently we are in the progress of further investigation for the problem and hope to have
some work to add to the dissertation with regards to the synchronisation of distributed patrolling between
the agents. We are also hoping to submit a paper we are writing alongside this project and the submission
deadline is around the same time as the dissertation deadline. For this project no changes from the original
project have been made and progress is as expected as laid out in the time plan presented in the specification.
For a more detailed analysis of progress to date a full Gantt Chart with marked progress can be found at
http://www.csc.liv.ac.uk/~m0tg/MSc.html

References

strategies. In Autonomous Agents and Multiagent System International Conference on Autonomous

mobile agents with distinct maximal speeds. Due to appear in ESA’11.

fence patrolling. In In AAMAS, 2008.


Tasks

- Location discovery of all participating agents.
- Even distribution of agents on the ring.
- Boundary Patrolling along the ring.

Model

- Synchronised model of a ring with anonymous agents randomly spaced along the circumference.
- No exchange of messages.
- No overpassing
- Maximal speed of 1

Observations

1. Periodicity. The distribution of the agents on the ring at time \( t = 1, 2, 3, \ldots \) is exactly the same as at time \( t = 0 \).

2. Rotation Index. Assume that \( n = n_C + n_A \), where \( n_C \) and \( n_A \) are the number of agents who move clockwise and anti-clockwise, respectively. The index of the rotation is given by \( i = n_C - n_A \), where we interpret \( i > 0 \) as a rotation by \( i \) positions in the clockwise direction and \( i < 0 \) as a rotation in the anti-clockwise direction.

3. Relative Primeness. If \( \gcd(|n_C - n_A|, n) = 1 \), then in time \( n \) all agents return to their original starting locations, having learned the relative starting positions of all other agents.

4. Odd \( n \). When \( n \) is odd, by repeating a simple randomized procedure sufficiently many times the agents can learn the relative starting locations of all other agents.

5. Even \( n \). For even values of \( n \), we will always have \( \gcd(|n_C - n_A|, n) \geq 2 \). So by repeating the randomized procedure, an agent can learn half of the starting positions by visiting them itself and can learn the other half by keeping track of the minimum time it takes to bump into each of its neighbours when at the positions it can find itself.
Time Complexity

- For odd $n$, the time complexity is $O(n^{3/2} \log n)$. In short we need one stage with rotation index -1 or 1, with high probability.
- For even $n$, the time complexity is $O(n^{3/2} \log^2 n)$. This time we require $O(\log n)$ stages with rotation index -1 or 1, with high probability.

Software

- Designed to aid understanding of the model and the location discovery problem and to detect patterns through analysis of positions of agents after each round.
- Verification of Observation 2, ($i = n_c - n_a$).
- Investigation of agents with variable speeds.
Further Work

- Algorithm adjustment for when n is even.
- Network Patrolling Solution
- One of the outcomes of this project will be a research paper to be submitted to STACS 2012.

Thank You
10.4 Appendix C - Original Time Plan
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MSc Project</td>
<td>79 days</td>
<td>Mon 06/06/11</td>
</tr>
<tr>
<td>2</td>
<td>Preliminary Discussions and background reading</td>
<td>11 days</td>
<td>Mon 06/06/11</td>
</tr>
<tr>
<td>3</td>
<td>Specification</td>
<td>11 days</td>
<td>Mon 20/06/11</td>
</tr>
<tr>
<td>4</td>
<td>Specification Draft 1</td>
<td>4 days</td>
<td>Mon 20/06/11</td>
</tr>
<tr>
<td>5</td>
<td>Specification Draft 2</td>
<td>3 days</td>
<td>Fri 24/06/11</td>
</tr>
<tr>
<td>6</td>
<td>Specification Draft 3</td>
<td>4 days</td>
<td>Wed 29/06/11</td>
</tr>
<tr>
<td>7</td>
<td>Specification Deadline</td>
<td>0 days</td>
<td>Mon 04/07/11</td>
</tr>
<tr>
<td>8</td>
<td>Design</td>
<td>19 days</td>
<td>Tue 05/07/11</td>
</tr>
<tr>
<td>9</td>
<td>Sumary</td>
<td>6 days</td>
<td>Tue 05/07/11</td>
</tr>
<tr>
<td>10</td>
<td>Design</td>
<td>6 days</td>
<td>Wed 13/07/11</td>
</tr>
<tr>
<td>11</td>
<td>Review</td>
<td>6 days</td>
<td>Fri 22/07/11</td>
</tr>
<tr>
<td>12</td>
<td>Presentation and Documentation</td>
<td>0 days</td>
<td>Fri 29/07/11</td>
</tr>
<tr>
<td>13</td>
<td>Software Implementation</td>
<td>42 days</td>
<td>Thu 09/06/11</td>
</tr>
<tr>
<td>14</td>
<td>Software v1.0 - Basic GUI</td>
<td>3 days</td>
<td>Thu 09/06/11</td>
</tr>
<tr>
<td>15</td>
<td>Software v1.1 - Animated GUI</td>
<td>4.5 days</td>
<td>Tue 14/06/11</td>
</tr>
<tr>
<td>16</td>
<td>Software v1.2 - Customisable no of Agents</td>
<td>11 days</td>
<td>Mon 20/06/11</td>
</tr>
<tr>
<td>17</td>
<td>Software v1.3 - Custom Speed for</td>
<td>6 days</td>
<td>Tue 05/07/11</td>
</tr>
<tr>
<td>18</td>
<td>Software v1.4 - Option of Agent Acceleration</td>
<td>5.5 days</td>
<td>Wed 13/07/11</td>
</tr>
<tr>
<td>19</td>
<td>Software v1.5 - Display previous time periods</td>
<td>6 days</td>
<td>Thu 21/07/11</td>
</tr>
<tr>
<td>20</td>
<td>Software v2.0</td>
<td>6 days</td>
<td>Fri 29/07/11</td>
</tr>
<tr>
<td>21</td>
<td>Experimentation and analysis</td>
<td>20.5 days</td>
<td>Mon 25/07/11</td>
</tr>
<tr>
<td>22</td>
<td>Dissertation</td>
<td>62 days</td>
<td>Wed 29/06/11</td>
</tr>
<tr>
<td>23</td>
<td>Abstract</td>
<td>4 days</td>
<td>Wed 29/06/11</td>
</tr>
<tr>
<td>24</td>
<td>Introduction</td>
<td>6 days</td>
<td>Tue 05/07/11</td>
</tr>
<tr>
<td>25</td>
<td>Background</td>
<td>5 days</td>
<td>Wed 13/07/11</td>
</tr>
<tr>
<td>26</td>
<td>Design</td>
<td>10 days</td>
<td>Wed 20/07/11</td>
</tr>
<tr>
<td>27</td>
<td>Realisation</td>
<td>9 days</td>
<td>Wed 03/08/11</td>
</tr>
<tr>
<td>28</td>
<td>Evaluation</td>
<td>9 days</td>
<td>Tue 16/08/11</td>
</tr>
<tr>
<td>29</td>
<td>Professional Issues</td>
<td>6 days</td>
<td>Mon 29/08/11</td>
</tr>
<tr>
<td>30</td>
<td>Conclusions</td>
<td>8 days</td>
<td>Tue 06/09/11</td>
</tr>
<tr>
<td>31</td>
<td>Bibliography/Appendici</td>
<td>5 days</td>
<td>Fri 16/09/11</td>
</tr>
<tr>
<td>32</td>
<td>Final Presentation</td>
<td>5 days</td>
<td>Mon 22/08/11</td>
</tr>
<tr>
<td>33</td>
<td>Prepare Presentation</td>
<td>5 days</td>
<td>Mon 22/08/11</td>
</tr>
<tr>
<td>34</td>
<td>Final Presentation</td>
<td>0 days</td>
<td>Fri 26/08/11</td>
</tr>
<tr>
<td>35</td>
<td>Dissertation Deadline</td>
<td>0 days</td>
<td>Thu 29/09/11</td>
</tr>
</tbody>
</table>