Simulating Metamorphism in a Swarm Robotics System

Final Report for CS39440 Major Project

Author: Mr. Rhydian Jenkins (rlj10@aber.ac.uk)

Supervisor: Dr. Elio Tuci (elt7@aber.ac.uk)

4th May 2016

Version 2.0 (Final Draft)

This report is submitted as partial fulfilment of a BSc degree in Computer Science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name [illegible] ……………………………………………

Date: 04/05/2016

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name [illegible] ……………………………………………

Date: 04/05/2016

Acknowledgements

I am grateful to my MMP tutor, Elio Tuci, for providing support and guidance throughout my project as well as ensuring that the project stayed on path. Elio was there to answer questions and provide alternative ideas to the structure of my project.

The Computer Science Department and the facilities of Aberystwyth University have been essential in allowing me to work on my project. Examples from previous students were available at reception, emails reminding me of approaching deadlines, and helpful forums and content on the Blackboard were available to me at all times.

I would also like to thank my lovely girlfriend, Alexandra, for proof reading my work and helping me keep my sanity during the project.

Abstract

Swarm Robotics is a new, scalable, and rugged solution to the co-ordination of multi-robot systems which allows each bot, in a potentially large system, to act purely out of self-calculated judgement based on its immediate environment. An overall manager could be used to individually feed instructions to each bot and micro-manage the entire task; however, this method falls flat when more and more bots are added or as the task becomes increasingly complex. No single machine can keep up with thousands of agents all needing detailed instructions at once.

Swarm robotics takes this problem and moves the responsibility of control over to every bot. With each bot thinking for itself, the system has no limit to the amount of agents that can be added. Not only does the system become more scalable, but also less prone to error. If any of the bots malfunction or if any unexpected conditions arise, the bots will have the ability to adapt and decide for themselves what to do, based on their observable environment.

My project focuses upon a specific problem within Swarm Robotics – getting multiple bots to metamorphose. This document will tackle all the problems faced with simulating S-Bots as they organise themselves into a specific shape. They will be doing this through the use of LEDs situated on the bot itself. The overall goal is to get multiple bots to organise themselves and connect with each other in order to form a desired shape.

This is a simulated project, which will not be tackling any of the hardware challenges such as image interpolation and hardware interfacing with the S-Bots. Although I will talk about implementing my software on read world S-Bots, we are simply trying to get our system working on simulated conditions.

Contents

1. Project Report Layout

This is a section by section breakdown on what was discussed in each part of the report.

1.1. Section 2: Background, Analysis, and Process

The first section will discuss the background research I conducted, where the topic of my dissertation came from, and how it evolved. This section will also be analysing current systems and looking at the state of Swarm Robotics in the modern industry, as well as what challenges the current systems are currently facing. Finally, the section will conclude by discussing about the process I will follow in the development of my system.

1.2. Section 3: Design

This section will begin by discussing the proposed specification. This is a list of all of the features my finished system will hopefully do. This includes a minimum list and a further list, which will be completed if time is available.

The architecture of the project will also be discusesd, involving where the files are and why they are there, as well as a detailed structure of the code involving a before and after UML diagram, before briefly talking about my choices in the GUI design.

Finally, I will justify the choice of the language and details of specific libraries I have implemented as well as talk about my choice to use a custom simulator.

1.3. Section 4: Implementation

This section will cover how I implemented each class in the program. There will be a lot of mathematical detail, as well as discussion on the values used in all of the equations and why I used them. We will also discuss the technical choices in algorithms will also be discussed within this section, as well as what else I could have used in their places and the overall roll of each class in the system.

1.4. Section 5: Testing

This section will focus upon my methods of testing, what I tested, and what I found whilst doing so; resulting in an evaluation of how successful testing was to the project. The major points of the testing have been split up into subsections where they will be discussed in further detail.

I will not discuss each and every single test case I conducted; instead I will simply highlight those which are important to the project success.

1.5. Section 6: Critical Evaluation

Section 6 will evaluate the overall success of the project. It will go into detail on both how the project could have been improved on as a whole, as well as what I think went well. I will also talk about what I would have done differently if I were to tackle a dissertation of this size again. The evaluation will be split up into subsections, each analysing specific parts of the project.

The section will then conclude with an evaluation of the project as a whole.

2. Background, Analysis, and Process

2.1. Background

Before even starting the research I knew that I wanted a project that seemed fun to do, interesting, and was something I have never done before. Ideally, something that was graphically heavy would have been ideal as that seems to be what I enjoy the most.

Swarm Robotics stood out to me as a potential project as it seemed like a new and exciting future technology which has only recently begun to be implemented in the laboratory [1]. Many papers have been published explaining the advantages of such a system.

I found that this kind of methodology is closer to many real world examples of nature where sometimes a vast numbers of agents – such as ants, bees, and birds - work together [2]. There is no leading project manager in an ants nest instructing each worker on how to do their job, neither is there any king starling that instructs every other starling on where to fly to form the amazing flocks that we see when the sun begins to set [25].

Images: http://www.lovethesepics.com/2012/11/sensational-starling-murmuration-far-out-flocking-phenomenon-37-pics-13-vids/ (right), http://www.alexanderwild.com/keyword/ant%20colony/ (left)

After finding out what the essence of what Swarm Robotics is, I began to read papers on the current state of robotics, discovering that the subject has a long way to go before it is truly relevant and prominent in every-day life. In other words, the theory of 'a thousand robots are better than one' is a solid idea however it seems that the implementation of such systems has yet to be fully achieved [29]. Systems such as AutoStore™[3] could potentially adopt a swarm-like approach to their machines; nevertheless, existing swarm robotics systems are by and large still limited to displaying simple proof-of-concept behaviours under laboratory conditions [27].

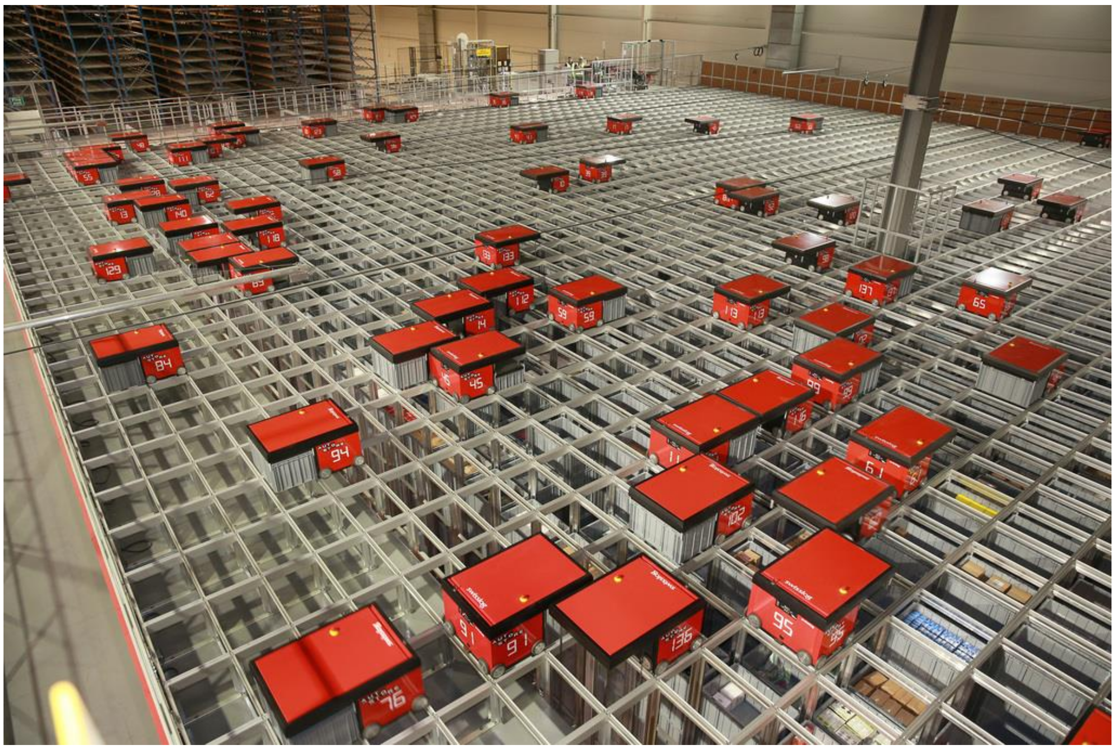


Image: AutoStore™ Warehouse

It did not take a lot of research to understand that the potential for such systems was forcefully driving the majority of research and innovation forward [26]. This formed the main reason for why Swarm Robotics was my chosen research topic - I see it as a technology that holds a large place in multi-robot systems of the future.

With further research, the lack of implementation became increasingly apparent. Getting a machine to decide on what to do in unknown conditions is always hard, and yet, these bots will need to do exactly that. Given a high level task instruction – such as “Build this”, “Search for survivors”, “Create this structure” etc. – the bots will have to decide how they are going to help complete this task using only the data in their environment that the sensors they are equipped with gather. In other words, labour allocation is unquestionably going to be one of the major challenges.

Finally, I found a topic which I would later invest as the focus of my dissertation – Metamorphism [20]. While looking through various articles and videos I found a paper discussing the need for bots to connect themselves together and form certain shapes in order to complete tasks that are too physically difficult for any single bot. Such tasks were situations, for example, where a gap in the terrain needed to be crossed without the use of a bridge, or where an object that is too heavy to move by a single bot is blocking the way.

Despite there already having been some development within this topic, it sounded like a fun and interesting topic, which would allow me to learn and understand the fundamental principles of Swarm Robotics.

2.2. Analysis

Currently Swarm Robotics has a huge variety of areas that need improving or developing before it is a viable option in the real world [4]. Some of the most challenging research questions are "How can the cooperative schemes inspired form the nature swarms integrate with the limited sensing and computing abilities for a desired swarm level behaviour?", and "How to design a swarm of robots with low cost and limited abilities which has the potential to show great swarm level intelligence through carefully designed cooperation?" [5]. Some of these problems are in need of cost effective and increasingly powerful hardware to become available, however, others need innovation in algorithms.

I found plenty of articles about ideas of Robots automatically building structures [6], exploring Mars [7], mapping unknown environments (e.g. buildings) [23][31], or scavenging for earthquake survivors [8]. However, most of these ideas are in very early stages of development and still have a long way to go before they are realised [24]. For now, the study of Swarm Robotics is limited to a relatively small group of robots preforming relatively simple and mundane tasks [9].

I took particular interest in sending swarms of robots to explore Mars. For such a task, the rovers will need to be able to traverse through rough terrain and overcome obstacles too treacherous for any single bot [10]. For example, a steep incline will mean that the bot could tip over. One of the solutions to this issue would be to have the bots attach themselves to each other in such a situation and tackle the obstacle as a group. For this to happen, the bots will need to independently arrange themselves to form a desired shape, which will involve the bots attaching themselves physically to another bot.

As such, I had my project. The problem was simple enough to understand; get the bots to form a particular shape. This task would have been easier if the system did not comply with the traditional rules and had centralised control telling each bot what, when, and how to do things. However, this would have defeated the point of Swarm Robotics. As previously discussed, this system would not be scalable in the real world and would not be able to function effectively in any and every environment it was placed in. Also, the bots will probably be small, cheap, and relatively simple mechanically [21]. This means that every bot in the system, which may potentially be thousands, will critically need to work correctly in order for the system to function; We cannot have 999 bots waiting for the 1000th to make a move before they all continue.

Essentially, a system that has no leader or dictator is needed. Every bot will need to be exactly the same, and every bot will need to be observant to its immediate environment and nothing else. If done correctly, an external input that has not been foreseen or pre-calculated could move the bots in a random direction, such as wind blowing bots off course [11], and it will not make any difference to the efficiency or effectiveness of the system outcome. During each stage of the simulation the bot will act on what it immediately sees and that alone; calculating exactly what to do on data that it has just gathered about its immediate environment.

Ideally, the bots will do this without fail and be ready to act on literally any environment they are placed in without the need of calibration. Such a system will not be prone to any faults in the environment (i.e. a blockage in the planned path) or in any other bots (i.e. if a bot breaks down). Such a system will also be easily scalable. If the system has too many bots, some can be taken away without worrying about the effect their loss may have on the rest of the system. If there are not enough bots, more can be added without the need for the others to even acknowledge the change in the amount of bots in the overall system.

In a perfect world, the bots will know the exact location of every other bot, as well as the exact location and properties of every rough patch of terrain and obstacle. As the environment of Mars is far from perfect, the bots will only be able to know the location and properties of their immediate environment. A lot of the time, the bots will not even be able to see any other bots and will appear to be completely on their own. This imposes a further question: how do you attempt to spread out, when you do not know the location of your fellow bots?

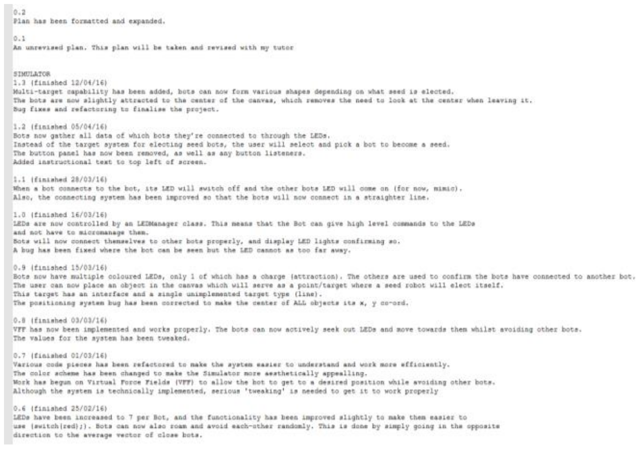
With all this taken into account, a feature list can now be constructed for the bots. They will need the ability to connect to other bots, as well as signal for bots to connect with themselves. They will also need the ability to sense their immediate environment, in addition to a method for communicating with additional bots. All of such abilities will need to be implemented without breaking the rules of Swarm Robotics.

These requirements immediately drew me to S-Bots [12]. They are small, cheap, and equipped with hooks and LEDs that can be used as a means for connection and communication. Conveniently, my university had several S-Bots in the lab in case I needed to test my system in the real world with physical bots. After further investigation, I was able to find articles that used S-Bots that attached themselves to other bots in order to lift them or move them [30].

2.3. Process

I knew that the planning stage was going to be particularly important, especially as I have never tackled anything of this size or complexity before. It was worth taking extra time at this early stage to ensure everything was effectively planned out, not only as this would help me later when writing the final report, but also as it would provide an effective guide to the project. Before planning any of the structure, code, or producing any UI designs, a full feature list was written. This feature list was a comprehensive outline containing all of the desired elements the finished project would contain and be able to do effectively. After the list was completed, each feature was broken down into individual tasks (Shown in appendices D.). For example, one of the simulators' features could be that the bots could be dragged and dropped by the mouse in order to move the bots around. This was considered quite an important feature and was to be imminently completed. However, careful planning of this revealed that other tasks, such as "produce a working GUI" and "Allow *something* to be drawn on the canvas", needed to be thought of and completed before this simulator feature could be implemented.

After thoroughly planning the tasks for each feature, the system could be chronologically planned out. More important features that needed to be implemented in order to start work on others were completed first. With a full list of features, versions were planned. A version was to be a milestone for testing and backups, meaning if something during the development was to go wrong, an earlier version could be revisited, and thus, the project could be started again from that earlier stage.



```
0.2
Plan has been formatted and expanded.

0.1
An unrevised plan. This plan will be taken and revised with my tutor


SIMULATOR
1.3 (finished 12/04/16)
Multi-target capability has been added, bots can now form various shapes depending on what seed is elected.
The bots are now slightly attracted to the center of the canvas, which removes the need to look at the center when leaving it.
Bug fixes and refactoring to finalise the project.

1.2 (finished 05/04/16)
Bots now gather all data of which bots they're connected to through the LEDs.
Instead of the target system for electing seed bots, the user will select and pick a bot to become a seed.
The button panel has now been removed, as well as any button listeners.
Added instructional text to top left of screen.

1.1 (finished 28/03/16)
When a bot connects to the bot, its LED will switch off and the other bots LED will come on (for now, mimic).
Also, the connecting system has been improved so that the bots will now connect in a straighter line.

1.0 (finished 16/03/16)
LEDs are now controlled by an LEDManager class. This means that the Bot can give high level commands to the LEDs
and not have to micromanage them.
Bots will now connect themselves to other bots properly, and display LED lights confirming so.
A bug has been fixed where the bot can be seen but the LED cannot as too far away.

0.9 (finished 15/03/16)
Bots now have multiple coloured LEDs, only 1 of which has a charge (attraction). The others are used to confirm the bots have connected to another bot.
The user can now place an object in the canvas which will serve as a point/target where a seed robot will elect itself.
This target has an interface and a single unimplemented target type (line).
The positioning system bug has been corrected to make the center of ALL objects its x, y co-ord.

0.8 (finished 03/03/16)
VFF has now been implemented and works properly. The bots can now actively seek out LEDs and move towards them whilst avoiding other bots.
The values for the system has been tweaked.

0.7 (finished 01/03/16)
Various code pieces has been refactored to make the system easier to understand and work more efficiently.
The color scheme has been changed to make the Simulator more aesthetically appealling.
Work has begun on Virtual Force Fields (VFF) to allow the bot to get to a desired position while avoiding other bots.
Although the system is technically implemented, serious 'tweaking' is needed to get it to work properly

0.6 (finished 25/02/16)
LEDs have been increased to 7 per Bot, and the functionality has been improved slightly to make them easier to
use (switch(red);). Bots can now also roam and avoid each-other randomly. This is done by simply going in the opposite
direction to the average vector of close bots.
```

Image: Versions document during the report state of my project

Each version of the simulator was saved into one directory, which was regularly backed-up, with a text document describing what was done and what was completed with each version; importantly, also noting when the version was started, and how long it took to complete. Any bugs found and fixed were also listed here, as well as any changes to the system that was made.

At this point the system was split up into versions, with each version having a list of features, and each feature having a list of tasks. Coding the project could now start at the beginning of the list. This method, while highly structured was also flexible, as it not only sparked more task ideas but also easily embraced this, allowing new tasks to be added to the list. By using this method I felt it was easy to see and note progress; you never felt that you were lost with what to do next.

Therefor the system was built on a mixture between version control and feature driven methodologies. I chose this method of development for a number of reasons. Firstly, it allowed me to clearly see progress and talk about the next steps with my tutor the whole way. Secondly, the regular intervals between versions allowed easy milestones for backups, acceptance tests, and refactoring. Finally, when it came to reporting on what I did, it meant I had a full comprehensive list of what I did and when I did it.

A lot of the testing was visual: “Did the bot move to where it was supposed to?”, “Do the bots look like they’re avoiding each other?”, “Does the bot move towards a target?”. These sorts of tests would have been very difficult to write code that checks if they’re done correctly. Therefore, this made a TDD approach unfavourable.

Another approach briefly considered was that of the Waterfall Model’s planned driven approach. Because I have never attempted to complete a project of this size and of this level of research, I decided that a more agile approach, with a more forgiving cost of change, would have been more appropriate.

3. Design

For projects as large as this one it is very important that the design is done correctly. Everything must be planned out at, least roughly, in order to give an idea of how all the pieces of the project will fit together, especially before any of the code can start. I found this beginning stage of the dissertation to be the most difficult and require the most time as I knew changes to this plan would be more costly the later I left them.

With this mentality I tried to have all of the classes and functions exactly planned out, before even starting, which would mean I could have simply written the code with very little worry about the overall architecture and how to fit it all together, leaving me to focus entirely on the functionality of the small section of code I would currently be writing.

This proved to be extremely difficult. With little experience of this method of design, and even littler experience of systems this large and complex, the original UML (2.2.1.) drastically changed as I write the project. This was was mainly due to both unforeseen issues with the design and more effective solutions to problems I encountered, which were only discovered after I started the code.

The idea to plan the entire project before starting it was ambitious as I have not attempted a project this large or complex. The plan was not followed and as a result, the design stage may easily be considered as a failure. However, I feel that having that basic idea of how things would fit together and building upon this foundation was far easier and more effective than simply 'making it up as I went along'.

3.1. Proposed Specification

I have to remember that this is a research project and that each feature will require more time reading, both about solutions and the backgrounds to those solutions, than actually implementing the code. With this in mind I feel the feature list produced was not only achievable but also challenging, which allowed opportunity for me to learn as much as I could on this project.

The final project will have a minimum ability to form a simple shape – the simplest of which will be a line. For this, the simulator will need to be complete with functionality for bots to move about freely and be able to sense their environment. These bots will need to have LEDs attached to them with the ability to change colour and also the awareness of other close surrounding agents within the simulation. These bots will also need to be able to position themselves so that they can connect with a desired bot, whilst avoiding the edges of the canvas and other bots. If two bots do end up too close to each other, they need to move away from those bots in a direction that will not move in the path of another bot. This process can prove to be extremely computational and needs to be efficient and effective to work properly in a simulation.

The system will also come with an intuitive GUI, which will allow me to add new, remove, and move the bots that are on the screen, possibly through a click-and-drag method, and elect a particular bot as a seed. The GUI will be intuitive and easy to use, as well as show clearly where the bots are, what direction they are facing, and what LEDs they currently have on.

Once the bots have the ability to roam, find a bot, and connect, I will now consider the system to be at its minimum specification. However, it is hoped that the final system will be inherently more sophisticated and feature-rich than this basic plan.

One of the further features will be the introduction of noise. In a real world simulation the bots' sensors will not be able to pin point the exact position and orientation of the LEDs that it sees. The camera that the S-Bots will be using will be prone to problems, such as dirt or smudges on the lenses, which will further increase the probability of slight differences in what the camera sees and what is actually there. This problem gets more serious the further away the objects it is seeing are. A simulator including this kind of distortion would give a more accurate behaviour of the bots, even though it will probably hinder the ability of the bots to make a shape accurately, quickly, or reliably.

A problem arises when getting a machine to look at an image and interpolate data from it. S-Bots are able to see their own LEDs with their camera and can sometimes interpolate it as a separate bot. Provided there is sufficient time once completing the essential feature list, this could potentially be a key feature I would like to focus further time upon. If such time is provided, I will hopefully give my bots the ability to detect themselves and remove their own LEDs from any further calculations.

Finally, a system that only makes a line seems like a uninteresting and unchallenging system to work on. I would hope that the final solution will be able to produce multiple shapes upon request. This would be useful in the real world as different shapes are useful for different things. For example, a hill too steep for the bots to traverse will need a *line* of bots to push each other up, however pushing a heavy obstacle out of the way of a path will require a shape that closer resembles a *block* of bots.

3.2. Overall Architecture

As mentioned, the early UML design, I had originally planned to follow, ended up looking very different to the final architecture of the project; this was as later in the project I found better ways to do things. One thing that didn't change was the tick/render system (taken from Markus Perssons *"MiniCraft"* [15]). The system works in a very typical game engine style; it implements the Runnable class and allocates each entity in the world with its thinking time, much like a CPU does to threads in a computer. This allocation of ticks is what gets the components in my simulator to '*think*'.

This kind of ticking system could be considered step based, where each bot will perform one action at each step, and no bot will perform another step until all other bots have completed theirs. However, it could just as easily be changed to a system where each bot preforms their step independently from every other, by splitting said bots into their own threads, and therefore, mimicking the real world more closely. As the bots are stateless and are not built on the assumption that every bot has completed a step before they complete theirs, I see no problem in implanting a step-based system in my simulator, as it can easily be converted if real bots ever need to be used.

Moving on to the GUI components, I decided to run them as JFrames simply because I utilised other parts of the java.awt.* library and mixing libraries is not conventional in the computing world. The same cannot be said for the BotCanvas class, which originally extended a JPanel component; however, this was later changed to a Canvas component as it had better rendering functionality. With this new component I was able to create a triple buffer strategy, which meant that the bots appeared to move at a much smoother motion and at a lesser processor cost.

A custom listener was used to listen for user actions. I found it more functional and simpler to have one class that extended all of the input listeners needed in the system, including the ActionListener, MouseListener, and ButtonListener components, than have separate classes for each of these. This CustomListener class has a pointer to a selected bot. If this bot is != null, it means that the user has selected the bot, by clicking and dragging it.

Overall, I feel that the final architecture is more logical and easier to implement that the originally proposed one. No system is perfect, so it is unfair to say that mine cannot be improved. For example, the Bot class is a very large. If further time were available, I would have somehow split this bot class up into further 'helper' classes like I did with the Sensors and the LEDManager. In the original UML, an Operation class was used to help the bot decide what to do next. This would have been a feasible addition to the Bot package, which would have aided in splitting the program up further, making it more decipherable.

3.3. Detailed Design

Note: These UML diagrams do not contain all of the methods in each class, but only the important and noteworthy ones. Basic methods such as getters and setters, or methods used to preform basic arithmetic such as turnAround() are not shown.

All UML diagrams were created using Creately.com, a free online diagram creation service.

3.3.1. Original Design UML

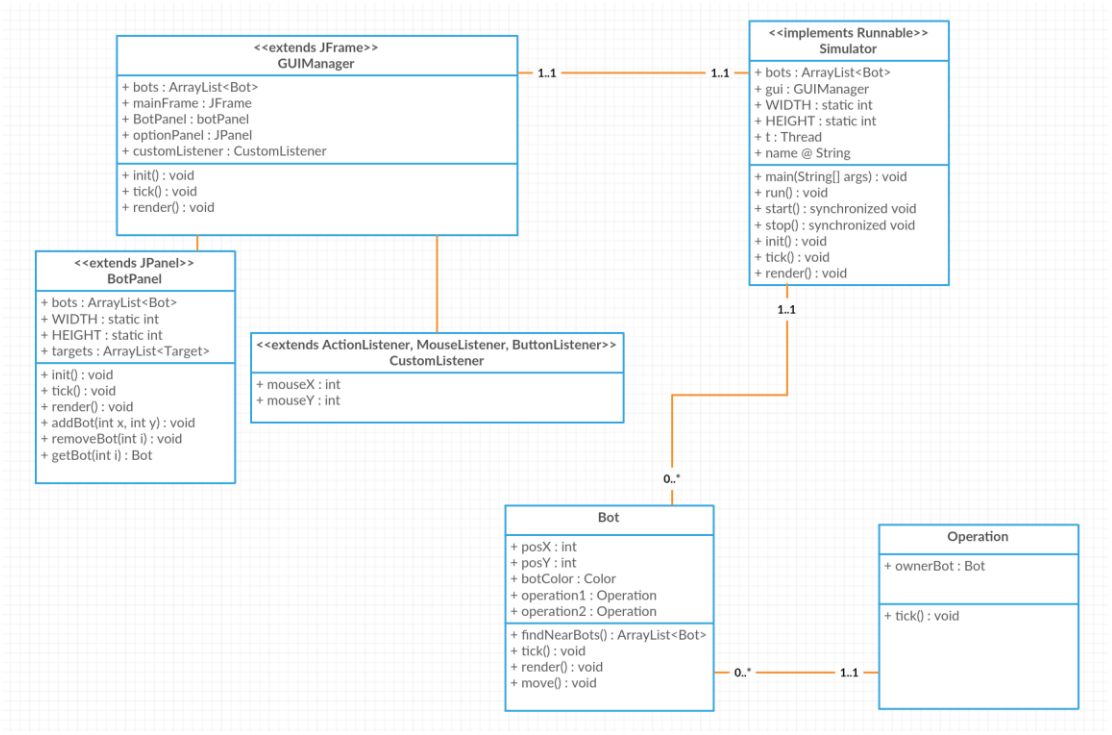


Image: Original UML design, created at Creately.com

3.3.2. Justification for Original UML

The entire architecture is ran on a 'tick / render' styled system. For every tick the class will 'think what to do next', and for every render the class will draw itself on the canvas. I have chosen this style of process because I have used it before in designing various games in Java, and thus, felt more confident in this style of approach.

The Simulator class (main) would be responsible for running the entire program. It would instruct all relevant objects to tick at set intervals, which would mean that the simulator would run at the same speed on all systems. The FPS or frames per second, is the number of renders the system can perform every second. Different systems with differing processing powers will vary in the FPS they are able to achieve, but will always simulate at a fixed or predetermined rate.

Bots are contained in an ArrayList owned by the GUIManager class. This class hands the ArrayList to the BotCanvas class, which allows the user to move the bots for testing purposes.

3.3.3. Final Design UML

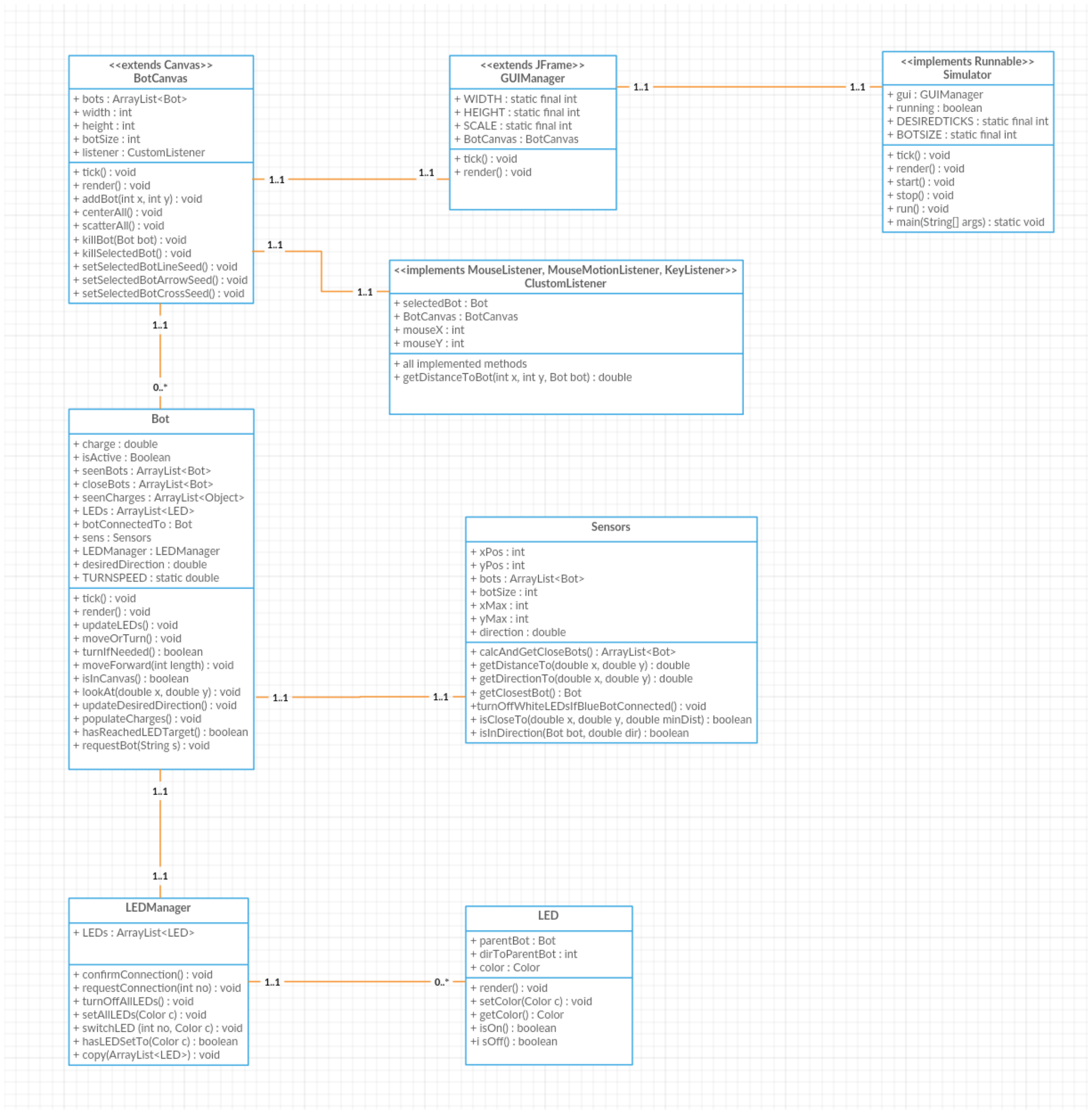

<<extends Canvas>>
BotCanvas
+ bots : ArrayList<Bot>
+ width : int
+ height : int
+ botSize : int
+ listener : CustomListener
+ tick() : void
+ render() : void
+ addBot(int x, int y) : void
+ centerAll() : void
+ scatterAll() : void
+ killBot(Bot bot) : void
+ killSelectedBot() : void
+ setSelectedBotLineSeed() : void
+ setSelectedBotArrowSeed() : void
+ setSelectedBotCrossSeed() : void
<<extends JFrame>>
GUIManager
+ WIDTH : static final int
+ HEIGHT : static final int
+ SCALE : static final int
+ BotCanvas : BotCanvas
+ tick() : void
+ render() : void
<<implements Runnable>>
Simulator
+ gui : GUIManager
+ running : boolean
+ DESIREDTICKS : static final int
+ BOTSIZE : static final int
+ tick() : void
+ render() : void
+ start() : void
+ stop() : void
+ run() : void
+ main(String[] args) : static void
<<implements MouseListener, MouseMotionListener, KeyListener>>
ClustomListener
+ selectedBot : Bot
+ BotCanvas : BotCanvas
+ mouseX : int
+ mouseY : int
+ all implemented methods
+ getDistanceToBot(int x, int y, Bot bot) : double
Bot
+ charge : double
+ isActive : Boolean
+ seenBots : ArrayList<Bot>
+ closeBots : ArrayList<Bot>
+ seenCharges : ArrayList<Object>
+ LEDs : ArrayList<LED>
+ botConnectedTo : Bot
+ sens : Sensors
+ LEDManager : LEDManager
+ desiredDirection : double
+ TURNSPEED : static double
+ tick() : void
+ render() : void
+ updateLEDs() : void
+ moveOrTurn() : void
+ turnIfNeeded() : boolean
+ moveForward(int length) : void
+ isInCanvas() : boolean
+ lookAt(double x, double y) : void
+ updateDesiredDirection() : void
+ populateCharges() : void
+ hasReachedLEDTarget() : boolean
+ requestBot(String s) : void
Sensors
+ xPos : int
+ yPos : int
+ bots : ArrayList<Bot>
+ botSize : int
+ xMax : int
+ yMax : int
+ direction : double
+ calcAndGetCloseBots() : ArrayList<Bot>
+ getDistanceTo(double x, double y) : double
+ getDirectionTo(double x, double y) : double
+ getClosestBot() : Bot
+turnOffWhiteLEDsIfBlueBotConnected() : void
+ isCloseTo(double x, double y, double minDist) : boolean
+ isInDirection(Bot bot, double dir) : boolean
LEDManager
+ LEDs : ArrayList<LED>
+ confirmConnection() : void
+ requestConnection(int no) : void
+ turnOffAllLEDs() : void
+ setAllLEDs(Color c) : void
+ switchLED (int no, Color c) : void
+ hasLEDSetTo(Color c) : boolean
+ copy(ArrayList<LED>) : void
LED
+ parentBot : Bot
+ dirToParentBot : int
+ color : Color
+ render() : void
+ setColor(Color c) : void
+ getColor() : Color
+ isOn() : boolean
+i sOff() : boolean
1..1
0..*

Image: Final UML design, created at Creately.com

3.3.4. Justification for Final UML

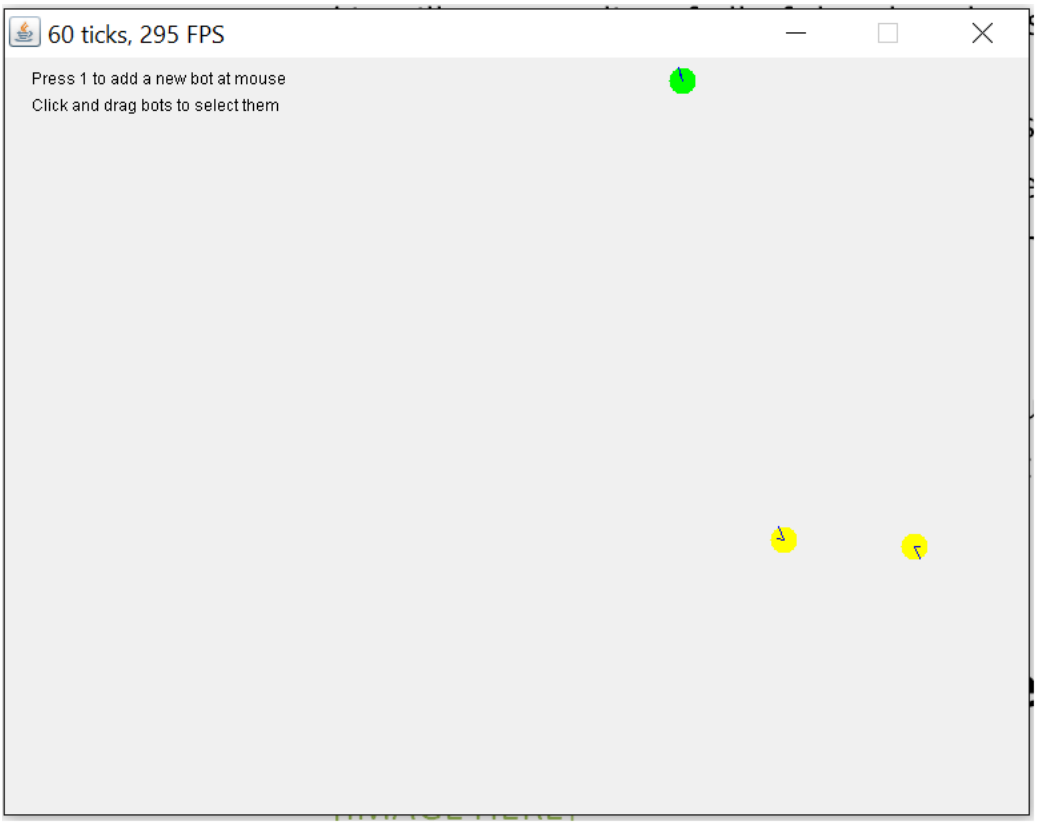
The final UML diagram holds many different components to the one originally designed. Firstly, the Operation classes have been removed, placing the burden of control within the bot class. This resulted in the bot deciding what it does, rather than the helper operation class. Secondly, each Bot class is equipped with a Sensors object, which is able to keep the Bot informed about its immediate environment. Methods such as getCloseBots() and getDistanceTo(int x, int y) are available; however, the class cannot tell the bot about entities outside of its range.

The sensors class contains an ArrayList with pointers to every bot on the canvas; breaching one of the primary rules of a Swarm Robotics system. This founding information is used to calculate which bots are in a given rage and compile them into a separate ArrayList to return to the corresponding bot object at every tick. This is used within the present system to simulate the data that would have been available from the camera, via fancy image interpolation algorithms.

This way of splitting the Bot class and the Sensors class would have been a very easy way to simulate noise, simply by getting the Sensors class to return an offset in the values that it provides. I feel this is a far more encapsulated way of allowing the bots to sense their environment.

Another major change is that the LEDs have been moved to their own class and controlled through an LEDManager. This gives high level command functionality to the bot over its LEDs; meaning it doesn't have to micro manage each LED when making a manoeuvre, rather it simply has to tell the GUIManager object the aim of its actions.

3.4. User Interface Design

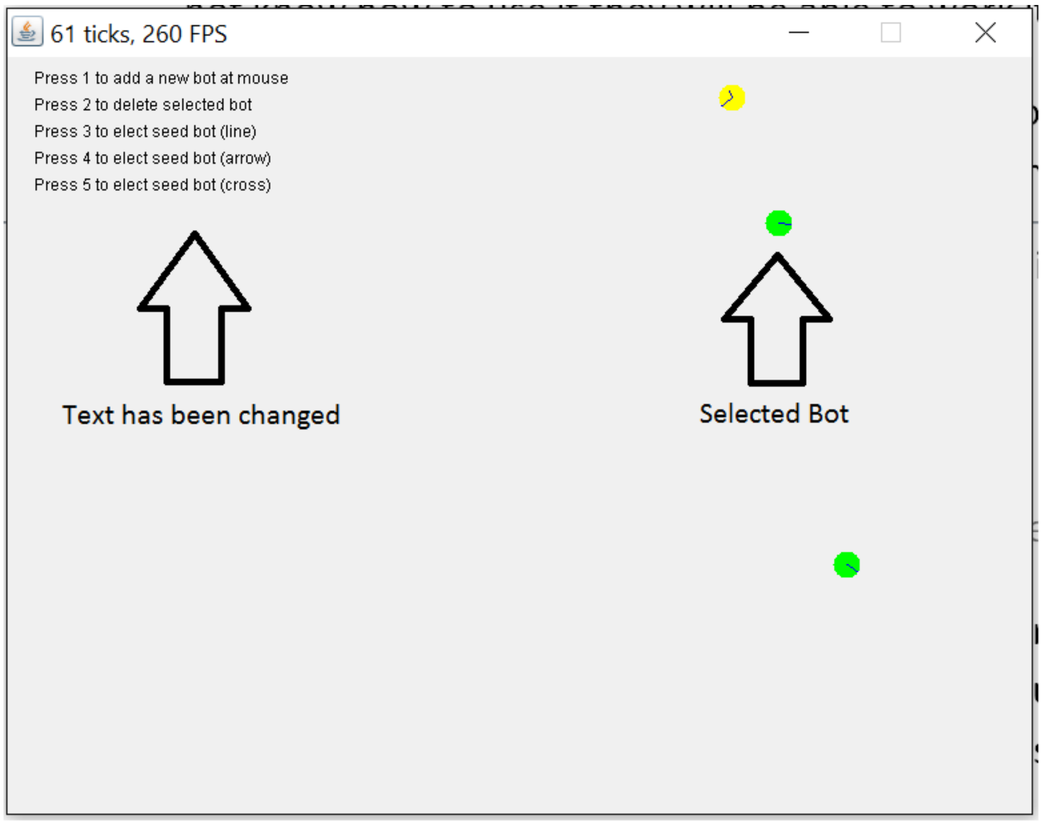

60 ticks, 295 FPS

Press 1 to add a new bot at mouse
Click and drag bots to select them

Image: Screen shot from version 1.3 (final) of my project, containing three unconnected bots

As the system was never intended for use by many people, but rather as a research tool, the user interface was not as important as a system, such as a phone app. However, I am still happy with the way that my system looks and feel that if it were to be put into the hands of someone who does not know how to use it they would intuitively know how to use it. Overall, I feel that the controls of the system are simple, clear, and user-friendly enough for the intended use of the system.

The instructional text seen in the top left corner of the window, changes depending on whether or not the user has selected a bot; this is as it only shows actions that the user can perform. When a bot has been selected (clicking and holding on it), text appears instructing the player on how to remove that bot or set it as a seed.


61 ticks, 260 FPS

Press 1 to add a new bot at mouse
Press 2 to delete selected bot
Press 3 to elect seed bot (line)
Press 4 to elect seed bot (arrow)
Press 5 to elect seed bot (cross)

Image: Text change when a bot has been selected.

During development, there was a panel at the bottom containing buttons for testing. These buttons included functionality to move the bots, such as centre all and scatter all, spawn in new bots, and remove bots already on the canvas. After most of the program was implemented, the button panel was removed and most of the functionality was transferred to the mouse. Text was superimposed to the top left of the canvas instructing the user on how to use the system, which seemed to make the user interface simpler.

It is essential that the window will not be resizable as the edges of the canvas act as the boundaries of the bots, and need to be known on compilation. To resize the window, a scale variable was implemented that was multiplied with the aspect ratio when the program was compiled. This means changing the size of the window is restricted to the developers, as access to this is via editing a variable in the code, resulting in the functionality not being available to the users.

The colour of the bot, while not a factor in bot interactions, was implemented so to enable the observer to visualise not only which bots have sited other bots but also those which are in close range to another bot. Three colours were used to suggest this to the user. Green rendered bots, infer that the bot cannot see anything in its environment - it has no information on anything in the canvas and assumes its alone and roams randomly until something is found. As soon as the bot discovers another bot the colour of the body is changed so that it renders yellow, assuring the user that I has indeed found at least one other bot. Finally, when the bot is rendered red, it means that there is one or more bots close enough for a connection.

3.5. Choice of Language and Libraries

Prebuilt simulators such as bullet could have been used to simulate the project rather than designing my own [13]. Although these pre-build simulators could have been adapted to my needs, the time and effort it would have taken to learn the simulators and adapt them would not be much different to building my own. Also, by going through the entire process and coding everything myself, I was able to get a much fuller understanding of my system and was able to learn some of the graphical algorithms that would have otherwise be given to me.

Moving on to the specific libraries, using the Java Swing library seemed the obvious choice, especially considering I have previous experience in their implementation, due to a modules focusing upon them during the second year of my degree.

C++ was a considered candidate language I could have used within the project. By using C++ I would have had further control over the pointers and memory management of the system, which would have possibly made the simulator run smoother and more efficiently. However, after further consideration, system performance was not an essential goal of the project, especially when considering the aim of the simulator was as a research tool. When taking this in to account, it was decided that the system could run sufficiently smooth for its desired function, built upon on a Java Platform. By selecting Java, arguably this will make the final application more interoperable as more machines have access to Java compilers than C++ compilers.

4. Implementation

As the system was further implemented, it became clear that the original UML design produced (see 2.2.1.) was in need of improvement. Thus, as the system development progressed the design slowly changed until it became the final and current version (see 2.2.3.).

Certain aspects of the implementation could still be improved. For example, in order to keep all of the bots in the canvas, I simply have them face the centre when they reach the edge. As a consequence of this the bots struggle to make shapes close to the edges of the canvas. Another flaw of this would be that this affects the validity of my swarm robotics system. In the 'real' world, the bots will not know if they have left the edges of the area and will not know where the centre is even if they did leave the area.

Another bug I noticed while implementing the LEDs is that the bots would not align properly to form a straight line. This issue was fixed by making the bots aim for the position that was 1 radius away from LED, in the opposite direction to that LEDs bot. In other words, rather than aiming straight for the LED, the bots would 'get into position' before facing, and connecting, to its target bot

I explored the possibility of implementing a kinematics model [14] into my bots so that they could turn and move forward at the same time, however I found that this did work well with the VFF algorithm as the bots often need to turn completely around while there other bots extremely close by. Rather than implementing this algorithm, the bots are currently unable to move forward and turn at the same time. Instead, they will stop moving when they need to change the direction they are facing. While this makes the bots perform manoeuvres more slowly, it leaves less opportunity for error, making the system more reliable.

4.1. The Classes

Here I will briefly cover the responsibility of each class in the system, and describe how it fits into the overall architecture of the project.

4.1.1. Simulator

This is the main class, located in the Main package. It is responsible for running the entire simulator. The bulk of the work was taken from Notches' Ludum Dares entry "MiniCraft" [15], which allows the entire project to run at a given tick-rate, while maintaining the highest frame rate possible. It is worth mentioning that MiniCraft is open source.

4.1.2. GUIManager

The GUIManager is a relatively simple class located in the Gfx package. It is effectively the JFrame – or the 'window' – that contains the BotCanvas. It sets the size, scale, and properties of the window (i.e. if it's resizable or not).

4.1.3. BotCanvas

This is all of the bots are rendered to the screen. It is a canvas filling the GUI JFrame that allows everything to be drawn at a triple buffered strategy. Located in the Gfx package, the BotCanvas has functionality for the user to edit some of the bots' positions. When the CustomListener class reports a mouse action, the BotCanvas will set the appropriate bots' position to the cursor's position.

Other responsibilities include adding and removing bots, electing seeds, and rendering the instructional text in the top left of the window.

4.1.4. CustomListener

This object is responsible for listening and dealing with all user mouse actions and keyboard presses by implementing the MouseListener, MouseMotionListener, and KeyListener classes. When a key is typed, it will inform the BotCanvas on what to do respectful to what key was typed.

When the user clicks on the canvas, the closest bot within a range is selected and set to follow the mouse. This is how the 'click-and-drag' feature is implemented. When the mouse is released, the selected bot is 'placed' back onto the canvas at the cursors' position.

This class can be found in the Listeners package.

4.1.5. Bot

This is the main Bot class. It contains various other classes, such as 'Sensors', 'LEDManager', and a list of seven 'LED' classes. All these classes work together through the Bot class; to aid the bot in deciding what it should do next.

This class is found in the Bot package.

4.1.6. Sensors

Each and every bot owns a Sensor object. The object is responsible for simulating the hardware mounted on the bot that senses the environment. The bot class can request certain information from the sensor class regarding near by agents, such as directions and distances to bots, positions and orientations to those bots, and colours of their LEDs.

This class is found in the Bot package.

4.1.7. LEDManager

LEDManager is another class owned by the bot. This class provides high-level command functionality and removes the requirement for the bot to micro-manage each of its LEDs. By splitting off some of the code from the bot and putting it in separate classes, the bot class becomes smaller, simpler, and easier to read.

High level commands include ‘confirm connection’, ‘request bot on x LED’, and ‘turn off all LEDs’. When these commands are issued it is up to the LEDManager class to decide what LEDs should be set to what colour.

This class is found in the Bot package.

4.1.8. LED

Located in the Bot package, this class represents a single LED light that will be attached to the bot. Each bot will have seven of these objects. Although it is the responsibility of the LEDManager to tell the LED what to do, it is the LED class’ responsibility to render itself and remember what colour it currently is.

4.2. The Functions

4.2.1. Virtual Force Field (VFF)

I feel that the VFF algorithm deserves its own section because it has been a massive part to the implementation of my project; this algorithm proved to be extremely useful and solved a lot of the challenges with the functionality of the bots. With it, the bots can now avoid other bots while ultimately moving towards a desired point of contact. It also deals with the angle of approach beautifully, and even allows me to tweak the values making the bots distance of comfort to other bots increase or decrease [22].

For example, if I feel the bots are getting too close to each other, I can simply make the bots have a lower negative charge. If I wanted the bots to be more attracted to an LED, I could simply make that LED have a higher positive charge. The challenge is getting a good balance of them both; making bots move towards white LEDs whilst avoiding other bots at the same time.

The algorithm works by having objects in the world either ‘push’ or ‘pull’ you towards them. Objects to be avoided have a negative charge and therefore will ‘push’ you away, and objects with positive charges ‘pull’ you towards them. The farther from 0 the charge is the more the objects push or pull force will be.

Below is a visual representation of the direction of force in the VFF algorithm, showing a negatively charged obstacle (top-left), a positively charged target (top-right), bot a target and an obstacle (bottom-left), and a path that a bot would follow when there is an obstacle in the way (bottom-right).

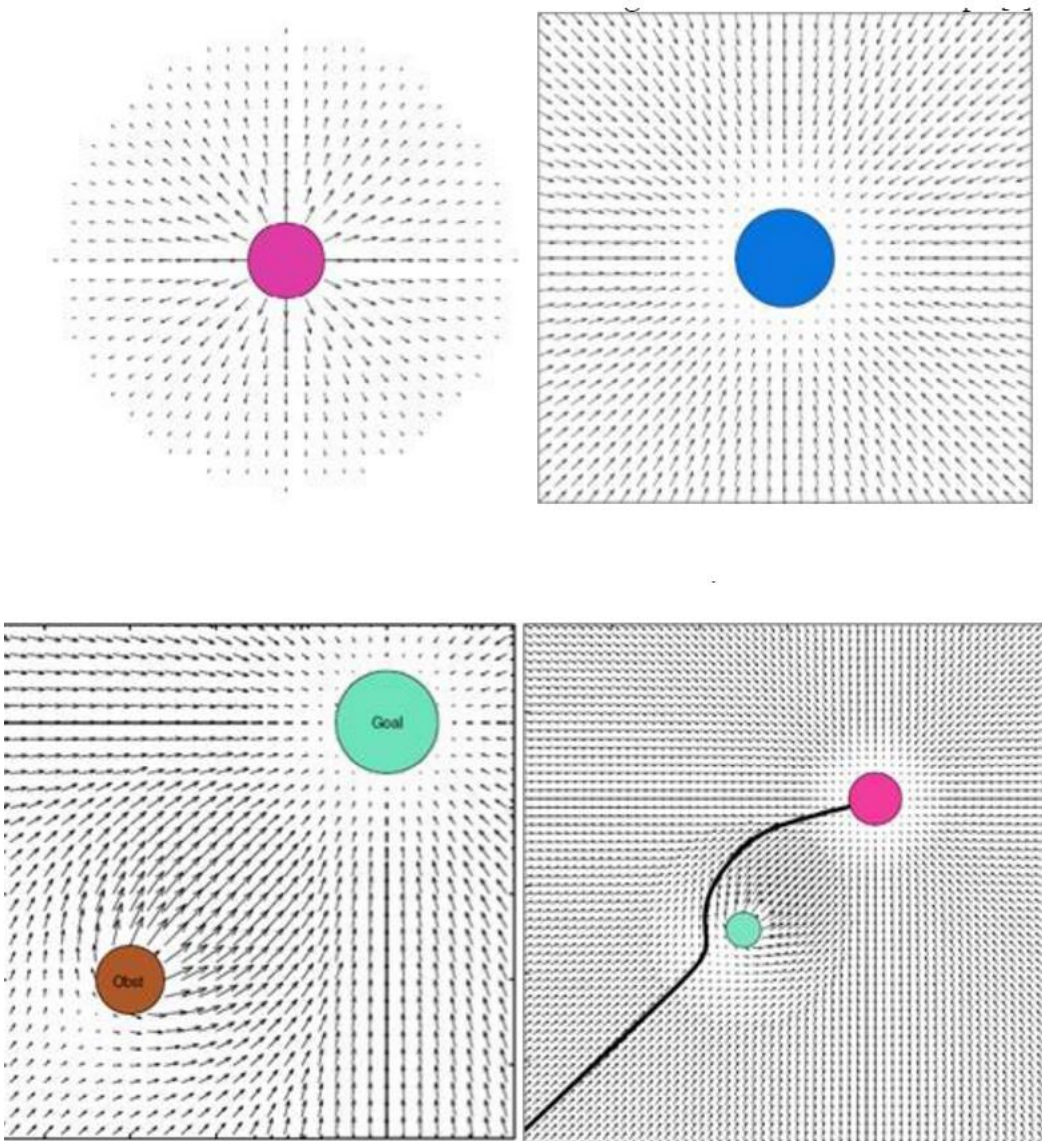


Image: VFF forces visualised, see [22]

Effectively, our bots are being pushed and pulled around the canvas. Another way of visualising these forces would be to look at a 3D gravity field representation (seen below) where repulsive, negatively charged, obstacles are represented by peaks in the field and attractive, positively charged, targets are represented as troughs. You can imagine the bots as spheres rolling around this field unable to make it up the slopes, whilst eventually falling into the target troughs. However, it is important to note that while the diagram below is not related to VFF's, it is effectively being utilised to aid in the illustration of a point. The values of the negative and positive charges are scaled the closer the bots gets – meaning the as the bot approaches the object its charge is amplified, becoming more repulsive or attractive.

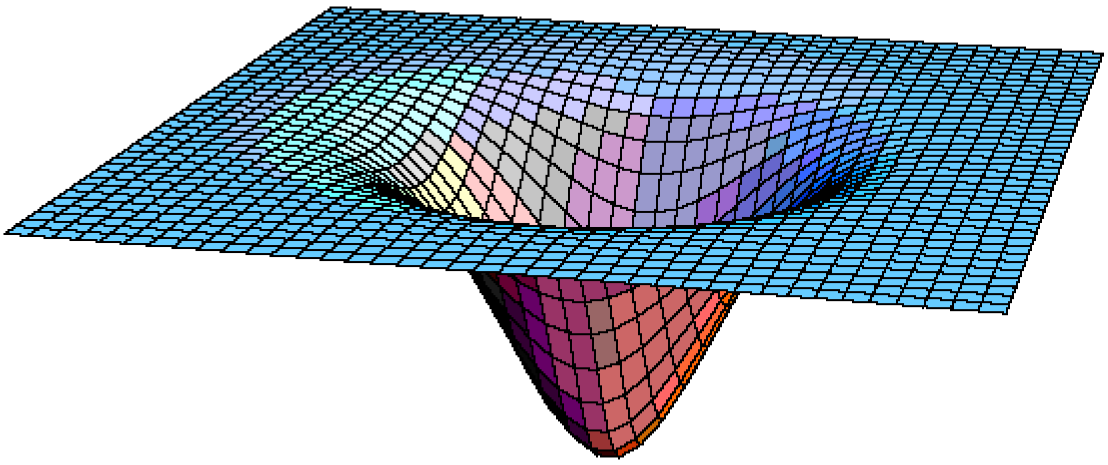


Image: Embedding diagram, http://www.upscale.utoronto.ca/PVB/Harrison/GenRel/GenRel.html

See Appendices C, Figure 9 for code snippet.

My system is not perfectly suited for a VFF algorithm [28] and there are issues to be addressed. For example, bots that have an infinite direction of movement (ones that can move any direction at any time and not just forwards) make use of the algorithm more effectively. Also, the algorithm tells our bots what speed they should be going. The higher the attraction to a target the faster the bot should be travelling. As our bots are fixed to a single speed and direction at any one time, it is easy to conclude that the bots do not take full advantage of the algorithms' features.

Another flaw is that our bots can be considered both a target and an obstacle. We want our bots to avoid other bots at all times, but also we want them to *sometimes* be attracted to a specific side of one, whilst avoiding the other side completely. This comes down to the adjustment of values until an effective medium is met.

Finally, the algorithm avoids space that is only just wide enough for our bots to traverse through. Sometimes, squeezing through a group of agents is essential when working in a system with a large number of bots, meaning the algorithm does not cope as efficiently as I would have liked under cramped conditions.

We want bots to be repulsive to other bots. We also want specific LEDs to be attractive enough to draw a bot in without causing a collision. This can be a problem as sometimes when larger structures form, the resultant negative force it is too much for a single LED to attract a bot for a connection, consequently resulting in the bots not connecting to larger shapes.

To allow bots to get closer to larger structures and connect to them we want the bots in that structure to be less repulsive than usual. I found by halving their repulsive charge, the bots would stay far enough away to avoid collision but get close enough to connect. So, the question is "how are we going to half the bots' negative charge when they're connected to a structure?".

When the bots have completed a connection, all 7 of their LEDs illuminate green. I gave each green LED a small positive charge that would result in an overall pulling force equal to half of the bots' pushing force. Thus, the charge for each green LED is as so:

$$LEDCharge^{green} = \frac{Bot^{charge} * 0.5}{LED_n}$$

Where:

$$LEDCharge^{green} = the\ charge\ of\ the\ LED\ (absolute\ value, must\ be\ positive)$$

$$Bot^{charge} = the\ absolute\ value\ charge\ of\ the\ bot$$

$$LED_n\ =\ the\ number\ of\ LEDs\ on\ a\ bot$$

4.2.2. Directions

4.2.2.1. The Global Direction System

A universal direction system was used in order to allow the sensors class to calculate directions relative to the bot. This global direction system can be thought of as a compass, but instead of facing 'north', you face '1'. The direction then rotates clockwise all the way around to 360 where it will then loop back around to 1. This means that instead of 'east' it's '90', and instead of 'south' it's '180', and instead of 'west' it's '270'. The direction is saved on the bot, and is initialled as a random number when the bot is created.

Some values that are calculated will be outside the range of 1 – 360. In these cases, Javas remainder operator (%) can be applied to normalise the value so that it once again lands in the desired range of 1 – 360.

4.2.2.2. Direction to a Bot

Directions are calculated using the following formula:

$$dir_{bot} = \frac{atan2(xPos_{relative}, yPos_{relative})}{\pi} * 180$$

Where:

$$dir_{bot} = direction\ to\ the\ bot$$

$$xPos_{relative} = relative\ x\ position\ to\ the\ bot$$

$$yPos_{relative} = relative\ y\ position\ to\ the\ bot$$

This formula will return a value between 1 and 360. As described (see section 4.2.2.1.), this number is used to represent the direction of where the bot is in relation to the bot calculating it.

Finding the direction to a specific bot was needed several of the algorithms used, including Virtual Force Field (VFF) calculations and confirmation of a new bot connection.

See Appendices C, Figure 1 for code snippet.

4.2.2.3. Average Direction Finding

Calculating the average direction, or the dot product, was originally used to avoid large groups of agents. The bot would set its target direction to the opposite direction of the calculated vector. This mean direction to bots was used throughout the development of the system up to the implementation of the VFF algorithm.

The following formula was adapted from one found on a public forum [16].

$$BT_X = \cos \sum_{n=0}^{n} \frac{BX_n * 2\pi}{360}$$

$$BT_Y = \sin \sum_{n=0}^{n} \frac{BY_n * 2\pi}{360}$$

$$angle_{avg} = \frac{\left(\frac{atan2(BT_Y, BT_X) * 360}{2}\right)}{\pi}$$

Where:

$$BT_X = Total\ relative\ X\ positions\ of\ the\ seen\ bots$$

$$BT_Y = Total\ relative\ Y\ positions\ of\ the\ seen\ bots$$

$$angle_{avg} = Average\ vector\ of\ seen\ bots$$

$$n = number\ of\ seen\ bots$$

See Appendices C, figure 2 for code snippet.

4.2.2.4. Turning to Desired Direction

Turning our bots to their desired directions is not instantaneous. Instead, each bot can only turn at an incremental rate per system tick. The bots calculate which direction, clockwise or counter-clockwise, is the shortest path before turning (see Appendices C, figure 3 for code snippet).

4.2.2.5. Moving in Direction Facing

How do we calculating the position one unit of travel in front of the bot. In the real world these values do not have to be pre-calculated and can simply be executed, however as our bots are simulated, positions need to be calculated. This is done by following the following formula:

Note: Math.toRadians() is built in Java method and is used instead of calculating the radians manually.

$$radians = \frac{\pi}{180} * Bot^{dir}$$

$$xPos^{next} = xPos^{current} + (\cos radians * length)$$

$$yPos^{next} = yPos^{current} + (\sin radians * length)$$

Where:

$$xPos^{current} = the\ current\ xPosition\ of\ the\ bot$$

$$yPos^{current} = the\ current\ yPosition\ of\ the\ bot$$

$$xPos^{next} = the\ xPosition\ of\ the\ bot\ after\ the\ move$$

$$yPos^{next} = the\ yPosition\ of\ the\ bot\ after\ the\ move$$

$$bot^{dir} = the\ direction\ that\ the\ bot\ is\ currently\ facing$$

$$length = distance\ of\ travel\ by\ the\ bot$$

See Appendices C, figure 4 for code snippet.

4.2.2.6. Looking at a [X, Y] Position

The VFF algorithm calculates an [X, Y] point on the canvas that the bot is being pushed. To execute, the bot must face that position [17]. The relative direction to that target is calculated like so:

$$dir^{target} = atan2(yPos^{rel}, xPos^{rel}) * \frac{180}{\pi}$$

Where:

$$dir^{target} = the\ directoin\ teh\ bot\ is\ aiming\ to\ face$$

$$yPos^{rel} = the\ relative\ yPosition\ from\ the\ bot\ to\ the\ target$$

$$xPos^{rel} = the\ relative\ xPosition\ from\ the\ bot\ to\ the\ target$$

See Appendices C, figure 5 for code snippet.

4.2.2.7. Calculating Distances

It is important for our bot to be able to tell the distance to something else on the canvas. This data is needed to calculate its VFF force. In the real world, this would be done by a clever image interpolation algorithm; however, for our simulated system, I will stick to relatively basic vector trigonometry. Pythagoras said it himself:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Where:

$$d = distance\ between\ (x_1, y_1)\ and\ (x_2, y_2)$$

See Appendices C, figure 8 for code snippet.

4.2.3. LEDs

4.2.3.1. Meaning of the LEDs

Each LED has 5 states; red, green, blue, white, and off. Each state has a different meanings and are used to signal different messages.

LEDs that are illuminated red are signalling that the bot has at least one connection still open. This is for bots that have already connected to it. A newly connected bot will turn blue confirming its connection until the bot it has connected to no longer have any red LEDs on, meaning it has confirmed all of its connections and is no longer waiting for any other bot to connect.

Green LEDs signal that a connection has been complete. A bot with green LEDs illuminated is part of a connected structure, and is no longer waiting for a bot to connect with itself.

Blue LEDs are used to signal a connection confirmation. When a bot connects with another bot, its LEDs will illuminate blue until the bot it has connected to confirm the connection using green LEDs.

White LEDs are targets for the bot. It indicates that a bot is needed to connect at the position of the white LED. If a bot is in need of another bot to connect to its rear, the back LED will illuminate white.

An LED that is off simply means none of the above. These LEDs will be completely ignored by other bots.

4.2.3.2. Updating the LEDs

The LEDs are calculated independently with each tick. That is, the previous sate of the bot does not get taken into account when updating the LEDs. Below is the pseudo code that is followed when deciding what colour the LEDs should be. This is done in the Bot class.

Firstly, if there are no close bots that are waiting for a bot to connect to it (i.e. no seen bots with white LEDs on), the bot will turn all of its LEDs off and continue roaming around.

Otherwise, if the bot has connected to another bot and it has not yet turned green, signal a connection by setting all LEDs to blue.

Otherwise, if we are waiting for a bot to connect to us and still have at least one white LED on, then close any connections that bots have attached to.

Otherwise, if our bot is in position to connect to another bot, save that other bot as the bot we are connected to and set our desired direction to face it.

Otherwise, if the bot we have connected to has its LEDs set to green and we have ours set to blue, it means the connection has been confirmed and we should request a new connection from behind by setting all LEDs to red and the rear LED to white.

See Appendices C, figure 6 for code snippet.

4.2.3.3. Closing the Open Connections

It can be said that the bot is waiting for a connection as long as there is at least one white LED on it. At every simulator tick that the bot has open connections, it checks its environment for a bot with blue LEDs on. If a bot is found and it is in a position to connect with one of its LEDs, it assumes that the connection has been complete and closes that connection. This is repeated until all white LEDs are turned off. Once all connections have been closed, the bot sets its LEDs to green, confirming to other bots that it has no more open connections.

See Appendices C, figure 7 for code snippet.

4.2.4. Rendering

4.2.4.1. Bots

Bots are rendered as circles on the canvas, and are drawn in such a way so that the centre of the circle indicates the bots' position. By taking the position as the centre and not the top left corner of the circle (as Java seems to prefer), it makes determining if the bots have collided with another bot easier. It can now be said that contact has been made between two bots if the distance between their positions is less than the diameter of a single bot.

The colour of the body does not play a part with communicating with other bots. I have simply used different rendering colours to indicate to the user how many bots are seen; agents that have green rendered bodies cannot see any other bot on the canvas, agents that have yellow rendered bodies can see at least one other bot, and agents rendered red can see at least one other bot that is close enough to connect.

Another property that is indeterminable to other bots is the desired direction. Close bots are unable to see where the bot is trying to face and only where the bot is currently facing. Users can see the desired direction of the bot by looking at the shorter of the two lines; the longer of which is the direction that the bots are currently facing, while the shorter line illistrates the desired direction of the bots.

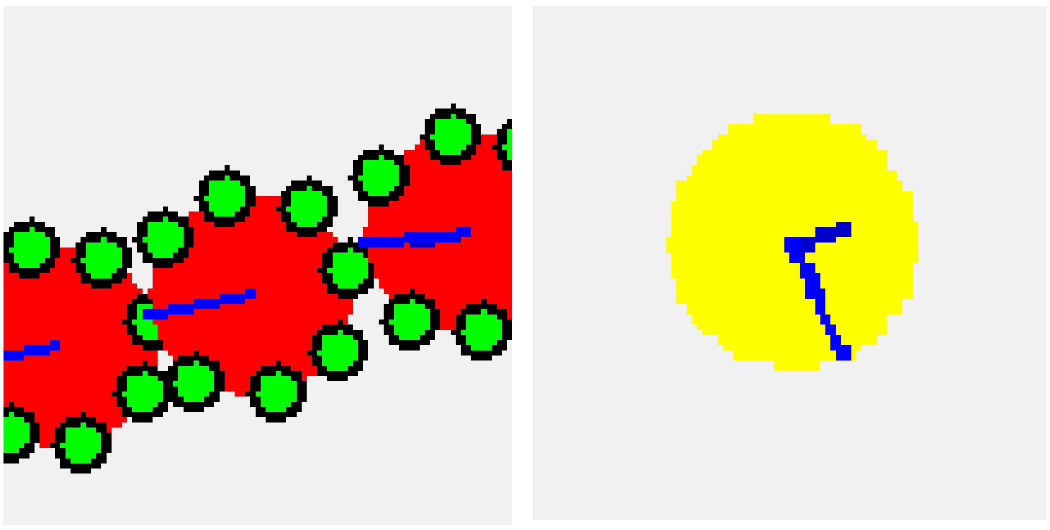


Image: Close up of the visual representation of some bots (as of version 1.3)

LEDs are small circles positioned on the outer edges of the bot with a fill colour if whatever the light that LED is currently illuminated. If the LED is off, nothing is rendered. Like bots, the LEDs position is taken from the centre of their circles.

4.3. The Behaviour of the Bots

The bots are stateless – that is, there is no need to save the bots previous position, orientation, or LED setup in order to determine what to do next. The only variable that is saved is whether the bot is connected to another bot. A system like this makes the bots far more rugged to unknown environments and events.

At the start of every tick cycle, the bots gather data on their surroundings. The sensors class will provide it with a list of all Bots that are within visual range of the bot. The bot then updates its LEDs (see 4.2.3.2.). After our LEDs are set, it is time for our bot to consider moving. If any of the LEDs are illuminated the bot will not move as we are already in a position that we want. If the bots direction is not facing its desired direction, the bot will turn.

Alternatively, if all of the LEDs are switched off, then we will continue roaming randomly, changing our direction every 100 ticks.

This process is repeated over and over again, under the condition that the bots' 'isActive' variable is set to true. When the user selects a bot, this variable will temporarily be set to false, rendering it inanimate so that the user can change the bots' position without it trying to move away.

4.4. Comparing with the Specification

Originally, the specification aimed for a system with a minimum functionality to form the most basic shape possible; a simple line. It left plenty of room for improvement including getting the bots to form multiple shapes, simulating noise, and more advanced algorithms for self-detection (see 3.1.). Although most of these further features were realised, there is still room for improvement in the system I have produced.

The simulator produced provides a clear GUI that allows me to clearly see the properties of the bots, such as their positions, LED states, and orientations. It also allows me to easily elect seed bots, move the bots around the canvas, add new bots, and remove existing ones.

I am happy with how the final system compares with the original specification. I was able to keep with the time schedule I set myself at the beginning and managed to complete *most* of the features I originally proposed. However, it is obvious to say that given more time on the project, further features would have been implemented. One such feature is noise, which could have possibly been implemented by including a slight random offset to the values the sensor class returns. The size of this offset will increase the further away the bot is.

By implementing noise I could have performed tests that determine how well my system can cope under certain levels of distortion. This would have allowed me to see the accuracy and reliability of my system under different levels of noise, which would have provided a minimum resolution the hardware in the real world would need to have in order to run my software.

5. Testing

5.1. Overall Approach to Testing

Testing for this project was mainly visual; most of the time, if the bots appeared to be working then they usually were. Despite this, I chose to have a numeric test for the time it took for shapes to form under different population densities (see section 5.3.).

There were also times that the bots appeared to work correctly but later found to be faulty. For example, when I implemented the functionality to detect the relative direction of a seen bot (see section 4.2.2.2.), it appeared to work correctly. It was only later when I discovered the direction would only calculate correctly when the bot was on the left side.

It seems my methods of testing were not flawless, however, as my system is in no way safety critical (a bot crashes? So what! Start the simulation again!), I do not regret my testing methodology. This is because I feel that the time I saved by only visually testing the different components of my code paid for the time spent fixing bugs that I would have caught if more detailed testing methods were used.It was only after the entire system was finished that I conducted detailed tests.

5.2. Stress Testing

It is important to remember that my system is simply a research tool and does not need to be able to cope under stress extremely well. If my code were to ever be properly implemented using real S-Bots, each agent will be thinking individually and independently. A large proportion of the simulation is calculating values that do not been to be calculated, such as the bots position after it has moved forward.

It is also important to remember that the system can lower its tick-rate and perform more slowly if it ever struggles to maintain constant thought.

That being said, many bots were added to the canvas after every completed version. This ensured that the frame rate could remain smooth at higher tick rates and under a heavy load of bots. I would add 300 bots, each ticking at 120 ticks per second, and ensure that the frame rate was higher than the tick rate.

A more important type of stress testing would be observing the bots' ability to form shapes correctly and efficiently in high population density environments; the first test was on collision detection without the presence of a seed bot. The test would involve measuring the length of time 300 bots took to reach a state where there are no collisions on the canvas. All bots were placed in the same position at the centre of the canvas, and were left alone until all bots were spread evenly. I feel my system performed extremely well and I am satisfied with the results of this test, which can be seen below.

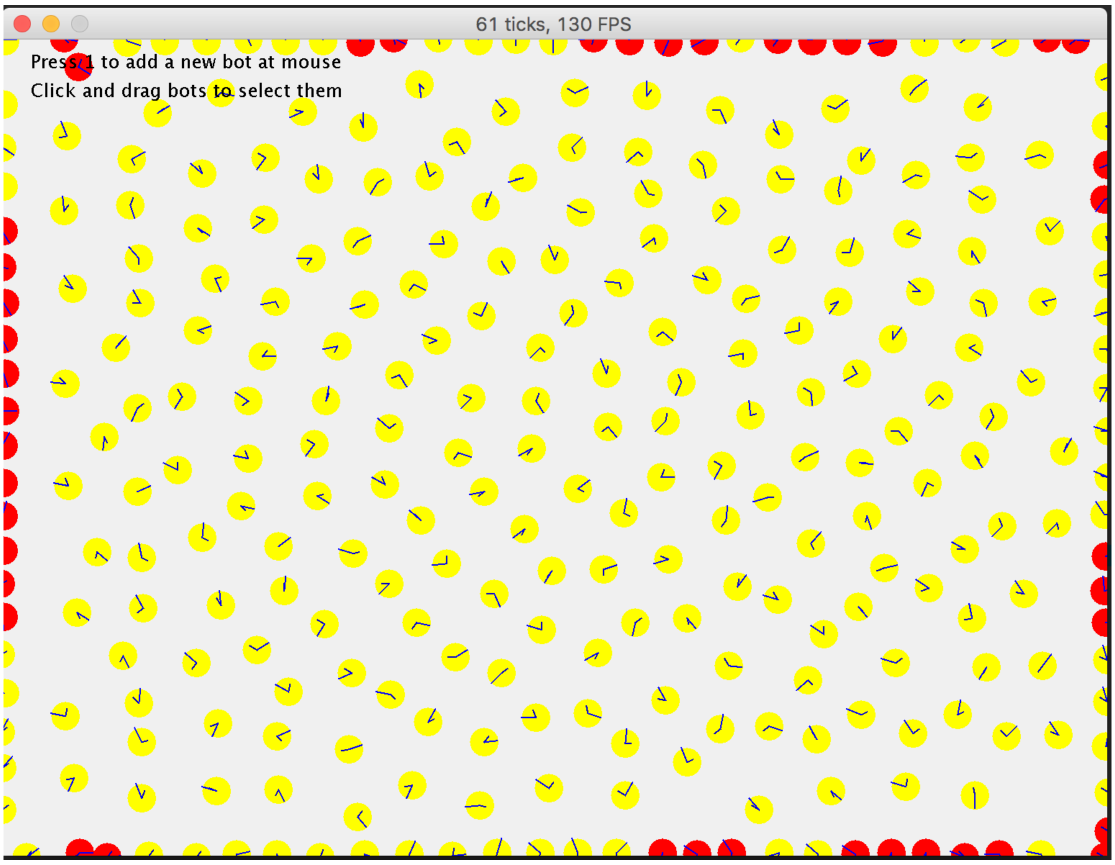

61 ticks, 130 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them

Image: 300 bots on the canvas with no LEDs on

You can probably see that the bots that come closest to colliding are those close to the edges of the canvas. As the system stands, when a bot leaves the canvas it is told explicitly, above everything else, to turn and face the centre of window. This feature of the simulator is an attempt to keep the bots in the bounds of the frame so that they remain in sight, and will not be present on systems implemented in the real world.

The same test was repeated with a larger canvas and 500 bots. The tick rate was lowered to 10 ticks per second to help the simulator keep up with the massive load.

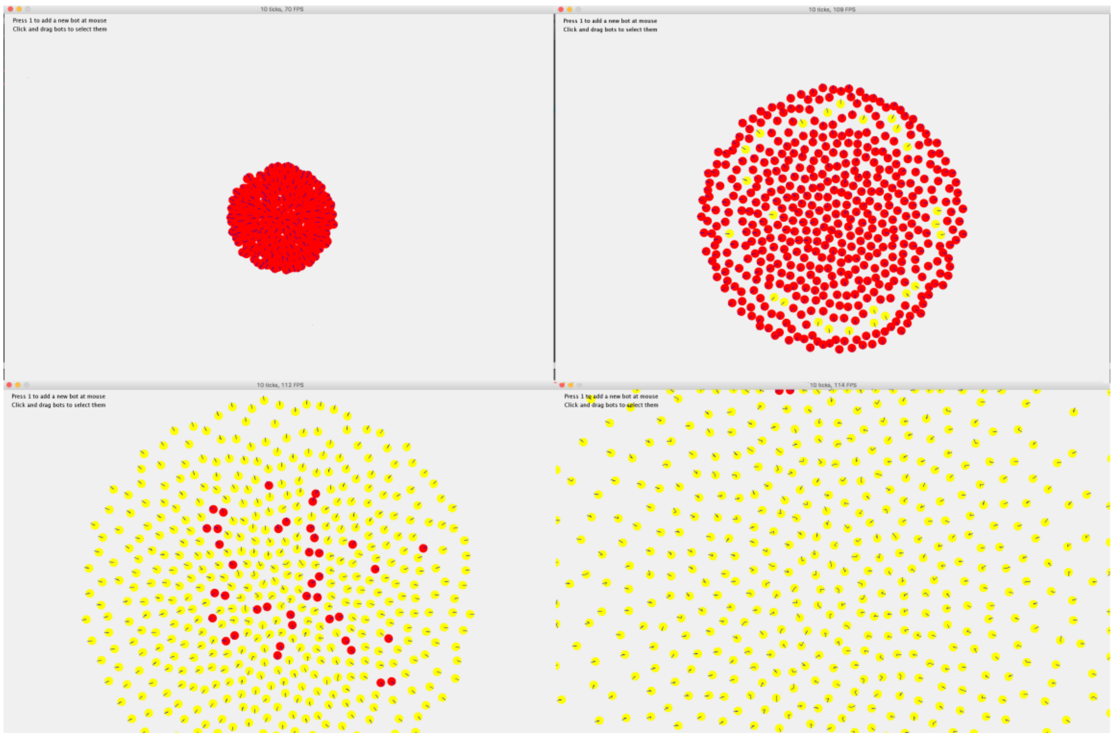

10 ticks, 70 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them
10 ticks, 109 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them
10 ticks, 112 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them
10 ticks, 114 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them

Image: 500 bots dispersing around the canvas at t = 1s, t = 5s, t = 10s, t = 15s.

The second test was of the bots' ability to adequately form a shape in high population density. Again, 300 bots running at 60 ticks per second were added to the canvas and a time was taken on how long it took for bots to connect with each other. Surprisingly, the rate of connection increases as the population becomes denser. Unfortunately, the VFF algorithm seemed to struggle with collision avoidance in these cramped conditions. Like gravity, the VFF force is amplified when bots get close to an LED, meaning they are pulled stronger when they are next to the LED. This amplified force can cancel out the repulsive force of bots. Consequently, I found that if the bots are very close to their target they would attempt a connection even if another bot is in the way. This is a rare phenomenon, which only happens in high density populations.

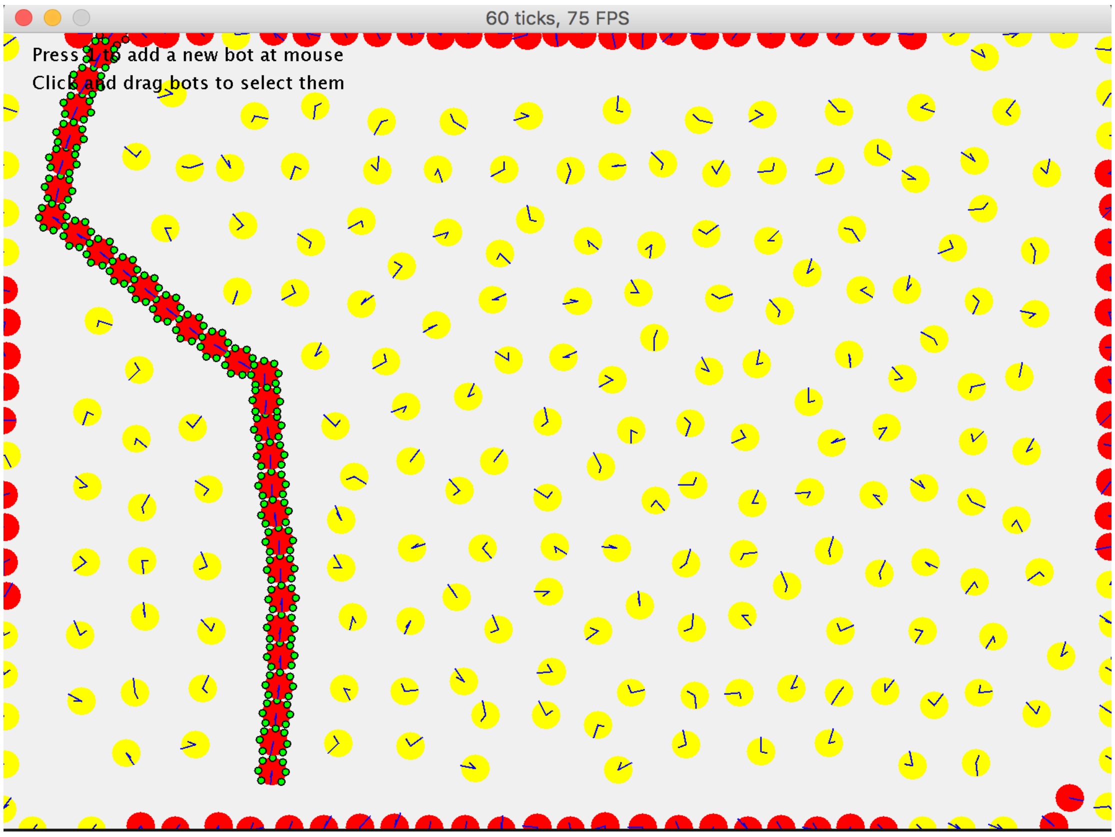

60 ticks, 75 FPS
Press 1 to add a new bot at mouse
Click and drag bots to select them

Image: 300 bots attempting to form a straight line under high population density

Obviously there is a fault somewhere. There are three bots at play during this stage; bot A, bot B, and bot C (see diagram below). Bot A is open for a connection when bot B gets in position. As bot B begins to face Bot A, Bot C is already close enough to connect with the currently turning bot B. Bot C then attaches itself to bot B before it has finished turning, which produces the skewed shape seen in the image above. To fix this issue, event orders need to change. Bot B must only illuminate its open connection once it is already facing bot A.

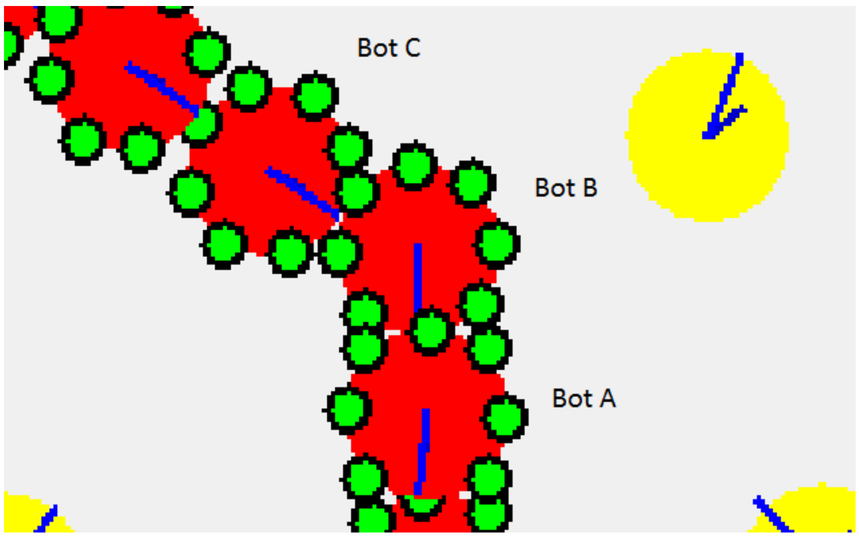


Image: Bot A, B, and C unsuccessfully connected to each other

5.3. VFF Testing

The VFF algorithm has several values that need adjusting in order to suit the system; the first of which is the charges of the bots and their respective LEDs. The bots' charge must be negative enough to keep other bots from colliding, whilst not being too negative to stop any bots being able to get close enough for a connection. This was certainly easier said than done.

Through trial and error, I found that by setting the push force of the bots' negative charge to less than half as strong as the pull force of the LEDs' positive charge I found the system works sufficiently well. The values used are -1.5 for the bots, and 4.0 for white LEDs.

Another problem arises when bots are connected to a large structure. When this happens, the push force of the combined bots becomes too strong for the pull force of a single white LED. To tackle this problem the bots need to be less reluctant to get close to other bots that are already in a connected structure (see 4.2.1.).

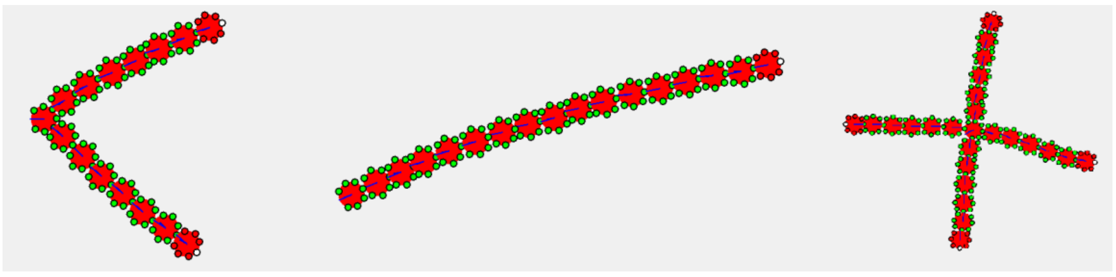
5.4. Acceptance Testing

After each version was completed, the project was compiled into a .jar and tested to see if the bots' behaviour was working correctly. After confirming the system passed the versions specification, I would deem the version 'complete'. A backup was made, and work begun on the next version.

I was not performance testing here; I did not measure how ***efficiently*** the features were functioning, only ***if*** they were functioning.

A small number of problems were indeed found when testing the systems performance; firstly, two bots with roughly equal distance to an LED would sometimes become stuck and appear to 'argue' over which one of them will move closer and take the connection. On the other side of the coin, if two bots are less than a diameters length to an LED, the VFF pulling force will be so high that they will both get drawn in and collide with each other. This is only seen to happen in highly dense populations of bots.

If these problems were to ever become unacceptable, I would look for a fix in the VFF algorithm – specifically the scaling of the pushing and pulling forces as the bot gets closer.

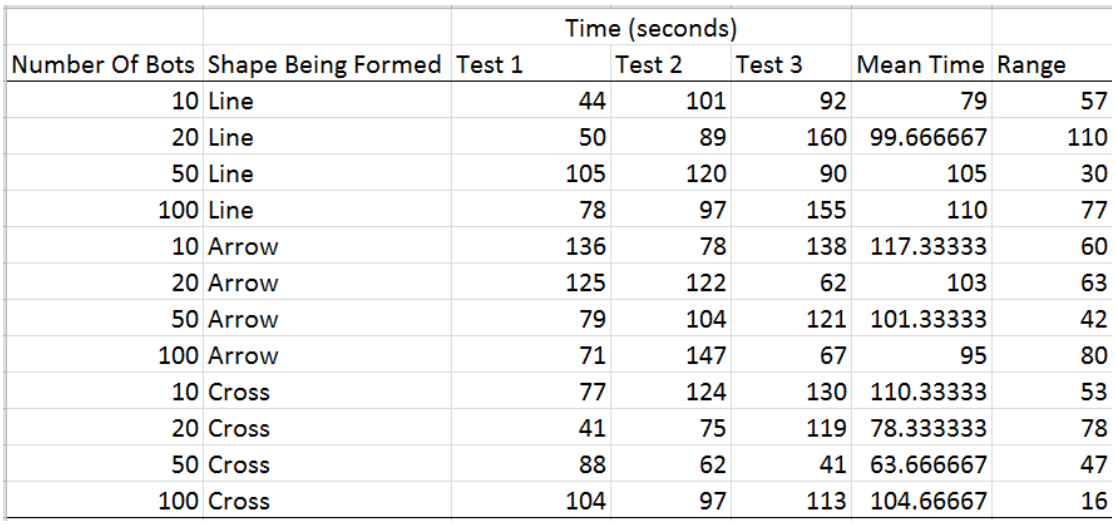


Images: Bots forming different shapes successfully (Arrow, Line, Cross)

5.5. Performance Testing

It is important to test not only if the bots do their jobs, but also how well they do them. To do this, I would populate the canvas with different amounts of bots and test to see how long certain shapes would take to form. Groups of agents that were able to form a shape faster were considered more successful in the tests. I am not considering *how straight* any structures produced to be, only recording the time they take to form.

Below is a table (screenshot from MS Excel) of the time it took for my system to produce certain shapes. I took three times of three separate tests. With this data, the mean time and range were calculated.



		Time (seconds)				
Number Of Bots	Shape Being Formed	Test 1	Test 2	Test 3	Mean Time	Range
10	Line	44	101	92	79	57
20	Line	50	89	160	99.666667	110
50	Line	105	120	90	105	30
100	Line	78	97	155	110	77
10	Arrow	136	78	138	117.33333	60
20	Arrow	125	122	62	103	63
50	Arrow	79	104	121	101.33333	42
100	Arrow	71	147	67	95	80
10	Cross	77	124	130	110.33333	53
20	Cross	41	75	119	78.333333	78
50	Cross	88	62	41	63.666667	47
100	Cross	104	97	113	104.66667	16

Surprisingly, no demonstrable time pattern emerges when changing either the number of bots in the system or the shape that the group is trying to form. The more bots on the canvas, the closer they must spread in order to stay away from each other. This means that the bots would tend to connect at a higher rate when there are more roaming on the canvas. The last bots to connect always take the longest to discover the shape, which may indicate work on the random roaming may need work if I ever wanted to increase the speed of the system.

6. Critical Evaluation

6.1. Research Evaluation

During early stages of design, I was able to find a lot of extremely helpful and relevant research regarding swarm robotics, and managed to learn a lot about Swarm Robotics as a whole, giving a huge variety of choice on what to focus the topic of my dissertation on.

Originally I had planned to implement a system that would simulate drones orbit a specified point in space while maintaining an equal distance to its neighbouring bots, however researching the problem and only managing to find a single paper [18] addressing it (and they cheated by using centralised control!), I decided the work load would have required more attention than a degree student could give it. My project was therefore simplified to having bots connect themselves to other bots, forming a shape.

The bulk of the research was conducted in the first stages of the project. A lot of papers were being read when I was still forming ideas on what my project would specifically be about. As the project matured, less research and more implementation was needed, meaning I was only reading a paper a week for a long period of time. These papers mostly consisted of solutions to implementation problems [19], as opposed to actual theoretical research into swarm robotics.

At this point I had written up a full feature list. I knew what my project was going to do; I just needed to figure out how it would be done. Despite this, I aimed to read at least one research paper on Swarm Robotics per week which discussed a proof of concept, or helped strengthen my knowledge in Swarm Robotics as a whole. I did this because the ultimate goal of this project was to expand my knowledge and understanding, which needs to be done in both practice and theory.

6.2. Specification Evaluation

I feel the specification written at the beginning of the project was both challenging and achievable. Each feature on the specification presented a technical challenge that almost always needed to be investigated further, allowing improvement in my programming, mathematics, and research skills.

Not only was a full feature list written, a full plan of the structure of the code was also attempted. It's easy to say that the code plan was unsuccessful as a lot of the structure was later changed in order to fit the implementation. However, I see it as an agile foundation that was a starting point susceptible to change when necessary. Having at least a small idea of how the code would be structured, while not finalised, helped me greatly. I will definitely attempt this level of planning again, despite how much I had to change the code.

The initial feature list produced was surprisingly complete. With the exception of noise, no new features were written during the project. Some were tweaked, such as the election process of seed bots where the user would manually elect a bot instead originally indeed process of finding a seed on the canvas. This process was changed as I wanted more control to the user over which bots become seeds and when.

Thus, I feel it can easily be said that I managed to correctly identify the features that my project needed to include before starting the code, helping the development greatly.

As already mentioned prior in the report, two specifications were written; the first of which was considered a ‘minimum’ set of specifications. If these features were not implemented the project would have deemed incomplete for failing to meet its initial intended purposes. The second list of specifications was only to be implemented once the first list had been complete. Not all features on this second list were ever intended to be completed, as the time needed to implement them was not expected to be available at the end. These features will instead only be implemented given the presence of spare time.

Unimplemented further features include more complex shapes and the presence of noise in the measuring instruments of the bots. These features do not directly affect the validity to my project, however they do give a more broad implementation to my system. More research would have been needed, requiring more time to complete.

It was planned so that each version of the project would take roughly equal time an effort to implement compared to the others. Some versions resulted in taking a lot more time than intended, while others took nowhere near as much time as I initially thought. This made the time difference of some of the versions massive. Longer versions meant longer times between refactoring and backups, as well as acceptance testing. Regular acceptance testing is an essential part of the development of any larger system. Without it, large portions of code are at risk of miss-implementation and may need to be replaced later in the project.

For example, one of the versions had a task to implement the bots’ ability to calculate the dot product of vectors to the bots that it sees as a stage of collision avoidance. After hours of working on this, the next version of the code had me implementing the VFF algorithm, rendering the majority of the code in the previous version obsolete. Because of this, I would put more time into thinking about the size of each version, and grouping the tasks in those versions together if I were to ever repeat this level of design again.

6.3. Process Evaluation

Although the methodology I used doesn’t tick all of the boxes, the project was planned out using a one similar to feature driven. Versions were planned out, and with each version a list of features were written. As I developed the system, any changes and bug fixes were logged in the text file containing all of the versions. This process has been described in further detail in section 6.2.

An agile methodology was definitely the best choice of methodology to follow. The amount of ‘on-the-fly’ modifications and redesigning that I needed to produce made any plan driven approach unfeasible. It was impossible to predict how long the implementation would have taken as I had never attempted anything to the level of complexity of most of the features, making it extremely difficult to predict how long it would take me to learn, plan, and implement the code.

Moving on to my time management; no sprints or version deadlines were used. My management of time rested entirely on working with the amount time I had left until my deadline and comparing it with how much I have managed to implement so far. If I felt I was behind, more hours were thrown at the project. Working with this, the problem always seemed to get fixed in an appropriate time frame. Although time management was not an issue with this particular project, my future work will have a more structured time schedule making it easier to determine if I am on schedule, and when I can expect to finish.

Admittedly, I was less organised with the report and simply decided to tackle the entire document after I had completely finished writing the program. I found that talking about the beginning of my project was difficult as I had it involved discussing something I did and how I did it that happened months prior.

Going through the process of creating a piece of software as large as this has taught me the importance of organisation. Future project reports will be written section by section as the content is completed, as well as having a well revised structure plan of the code.

6.4. Product Evaluation

While the VFF system worked well when one LED was on and managed a lot of the problems that needed tackling, it was definitely not perfect. For example, there are instances where bots would appear to 'argue' with each other over which one of them will take the plunge and connect to a white LED. This would occur most commonly when there are two bots at equal distance to the target. These bots are never permanently stuck, and always eventually settle their differences and elect one to take the connection.

Also, it was demonstrated in section 5.2. that the system does not cope well under stress. Bots would perform a lot faster in densely populated canvases, however, they become a lot less reliable and often collide or miss-connect.

Tweaking VFF values helped avoid these problems however the system still occasionally reveals a fault. Because of how rare these occurrences are, and that they only occur under cramped conditions, I accept these as a known issues and I am willing to release the final versions of my system with these issues.

As the system is more of a research tool and never intended for use by more than a handful of people, not much needs to be said about the GUI. I personally find it easy to use and clear to see what is currently happening in the simulation, which is more than enough to pass its acceptance test. Improvement could be made but it would simply be a waste of time.

The program itself is extremely interoperable as it can be used on any machine with access to a JVM. Also the file size is small enough to send in an email which makes it convenient to store.

All features along with most of the further features proposed in the specification were implemented. Though I am extremely happy with what I have managed to achieve in the time frame, I still regret the absence of correctly implemented noise in my system as such a feature would have been almost essential if I wanted to simulate my work on real S-Bots. By implementing this feature, I would have also been able to test my system under different levels of noise in order to determine the minimum resolution of accuracy that the hardware would need for my software to work reliably on real bots.

6.5. Conclusion

A lot has been said on potential ideas involving building, search and rescue, and planetary exploration, however, most of it has yet to be implemented due to a lack of innovation in the technology. These ideas remain as proof of concept in labs and simulators only.

My system only tackled a small and specific part of Swarm Robotics. The research I conducted and the system I produced has taught me a great amount including the importance of organisation when building software, time management skills, and the topic of Swarm Robotics itself.

I feel my project was successful in both a research and software engineering point of view. I managed to learn a lot about the growing industry of Swarm Robotics during the design, research, and implementation, and I feel my skills in mathematical algorithms and programming have been strengthened. The system was completed with only a few of the further features not implemented., and the features that were implemented underwent proper planning, testing, and documentation.

For future projects, I will be sure to allow as much time for planning the report as I do writing it, and be sure to document what I have done as I do it instead of leaving it all to the very end. As someone who struggles with getting their thoughts down on paper fluidly, I would say that the report turned out to be the most challenging part of the project for me.

I have extremely enjoyed myself throughout the development of this project and hope to have the opportunity to work on systems such as this one in the future.

Appendices

A. Third-Party Code and Libraries

Eclipse was used as an IDE for the development of my system.

Code from Markus Perssons “MiniCraft” [15] was used in the allocation of ticks and renders.

Websites such as StackOverflow.com (for syntax checking) and Creately.com (for diagram creation) were used throughout the development.

B. Ethics Submission

AU Status Undergraduate or PG Taught

Your aber.ac.uk email address rlj10@aber.ac.uk

Full Name Rhydian Jenkins

Please enter the name of the person responsible for reviewing your assessment. Elio Tuci

Please enter the aber.ac.uk email address of the person responsible for reviewing your application elt7@aber.ac.uk

Supervisor or Institute Director of Research Department cs

Module code (Only enter if you have been asked to do so)

Proposed Study Title Swarm Robotics

Proposed Start Date 02/2016

Proposed Completion Date 04/05/2016

Are you conducting a quantitative or qualitative research project? Mixed Methods

Does your research require external ethical approval under the Health Research Authority? No

Does your research involve animals? No

Are you completing this form for your own research? Yes

Does your research involve human participants? No

Institute IMPACS

Please provide a brief summary of your project (150 word max) Metamorphism in a Swarm Robotics system. I will simulate Robots organising themselves with no external commands to make shapes on a computer screen.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material? Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data? Yes

Does the research pose more than minimal and predictable risk to the researcher? Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to? No

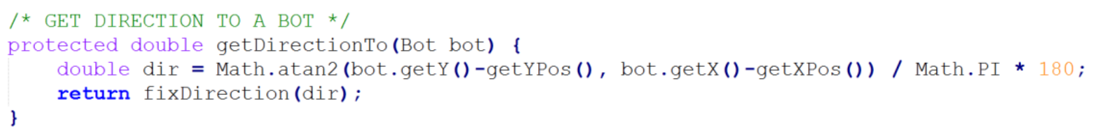
Please include any further relevant information for this section here: *null*

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one. Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change. Yes

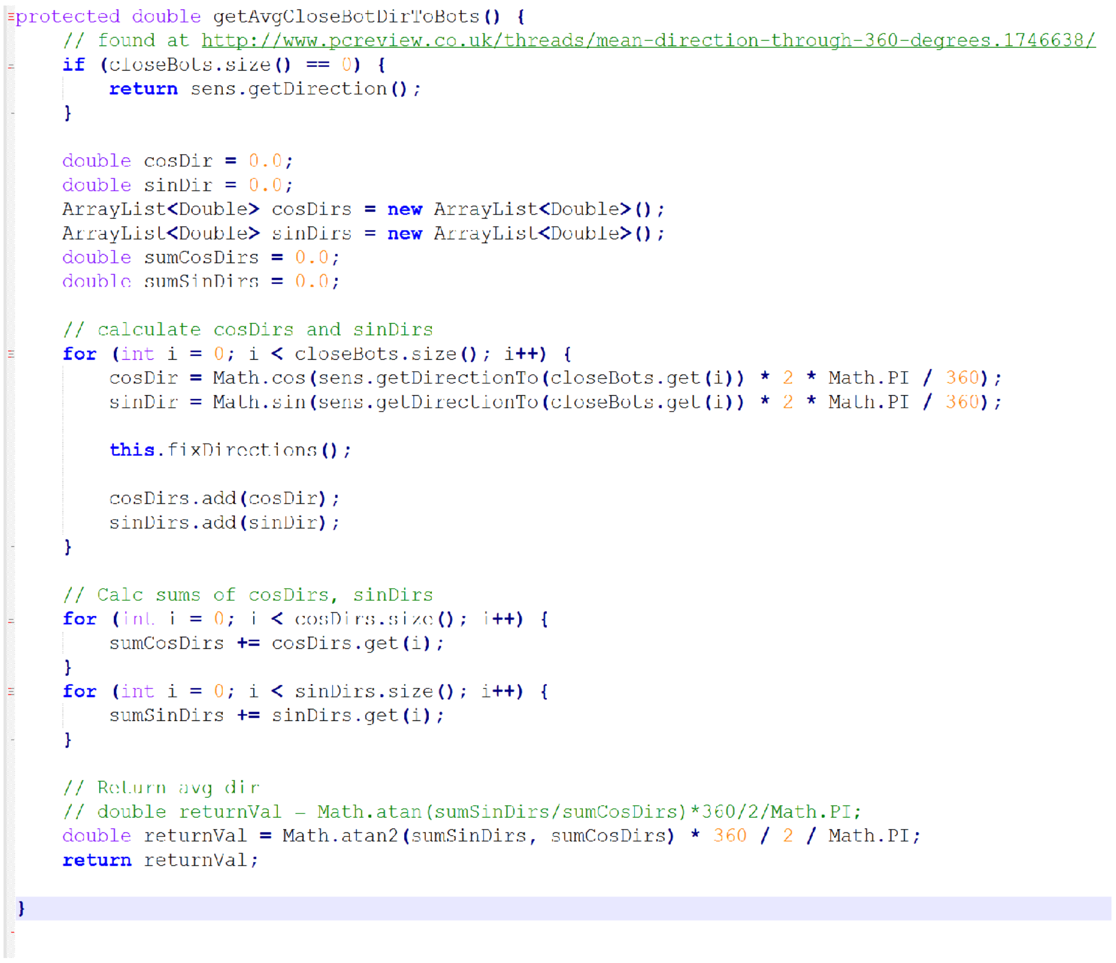
Please include any further relevant information for this section here: *null*

C. Code Samples



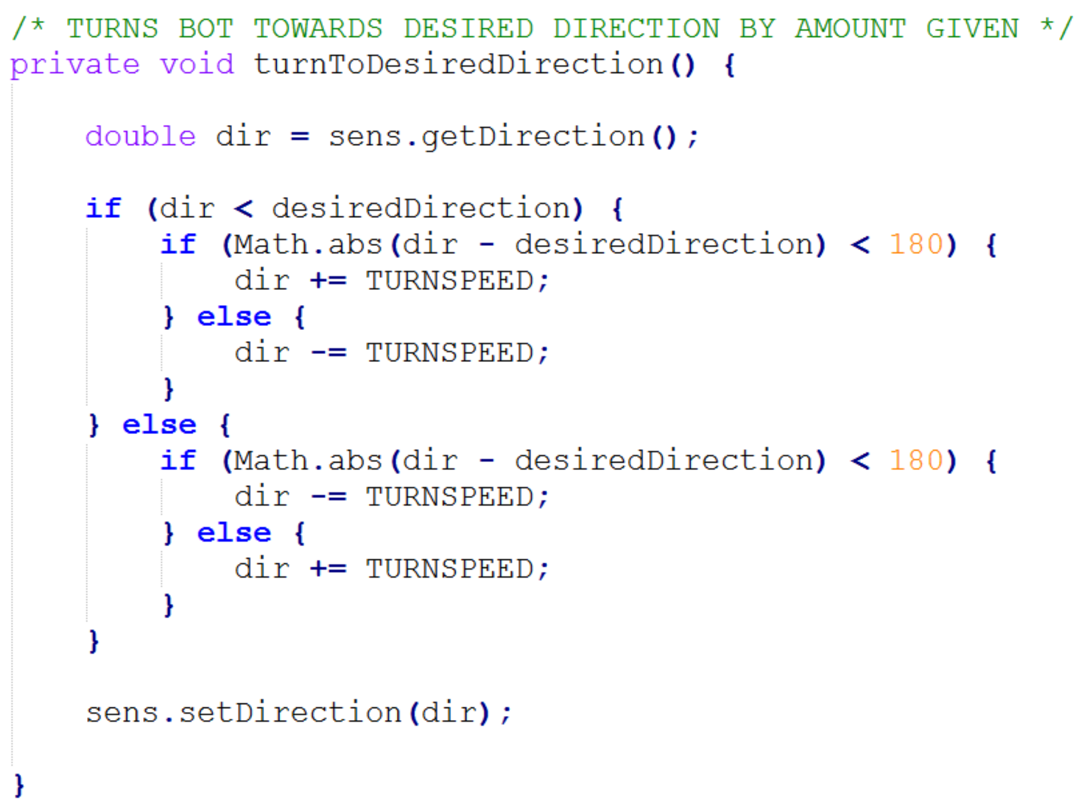
```java
/* GET DIRECTION TO A BOT */
protected double getDirectionTo(Bot bot) {
    double dir = Math.atan2(bot.getY()-getYPos(), bot.getX()-getXPos()) / Math.PI * 180;
    return fixDirection(dir);
}
```

Figure 1. Finding the direction to a bot.



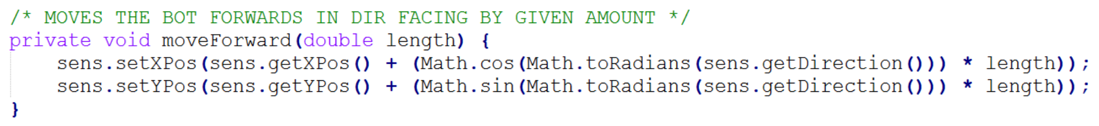
```java
protected double getAvgCloseBotDirToBots() {
    // found at http://www.pcreview.co.uk/threads/mean-direction-through-360-degrees.1746638/
    if (closeBots.size() == 0) {
        return sens.getDirection();
    }

    double cosDir = 0.0;
    double sinDir = 0.0;
    ArrayList<Double> cosDirs = new ArrayList<Double>();
    ArrayList<Double> sinDirs = new ArrayList<Double>();
    double sumCosDirs = 0.0;
    double sumSinDirs = 0.0;

    // calculate cosDirs and sinDirs
    for (int i = 0; i < closeBots.size(); i++) {
        cosDir = Math.cos(sens.getDirectionTo(closeBots.get(i)) * 2 * Math.PI / 360);
        sinDir = Math.sin(sens.getDirectionTo(closeBots.get(i)) * 2 * Math.PI / 360);

        this.fixDirections();

        cosDirs.add(cosDir);
        sinDirs.add(sinDir);
    }

    // Calc sums of cosDirs, sinDirs
    for (int i = 0; i < cosDirs.size(); i++) {
        sumCosDirs += cosDirs.get(i);
    }
    for (int i = 0; i < sinDirs.size(); i++) {
        sumSinDirs += sinDirs.get(i);
    }

    // Return avg dir
    // double returnVal = Math.atan(sumSinDirs/sumCosDirs)*360/2/Math.PI;
    double returnVal = Math.atan2(sumSinDirs, sumCosDirs) * 360 / 2 / Math.PI;
    return returnVal;

}
```

Figure 2. Calculating the average vector (dot product) of all close bots.



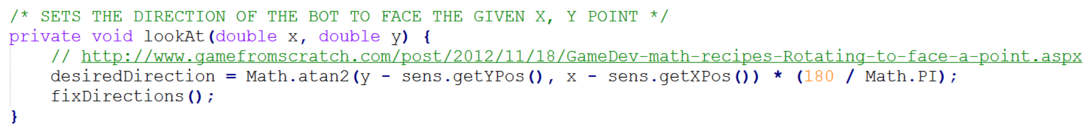
```java
/* TURNS BOT TOWARDS DESIRED DIRECTION BY AMOUNT GIVEN */
private void turnToDesiredDirection() {

    double dir = sens.getDirection();

    if (dir < desiredDirection) {
        if (Math.abs(dir - desiredDirection) < 180) {
            dir += TURNSPEED;
        } else {
            dir -= TURNSPEED;
        }
    } else {
        if (Math.abs(dir - desiredDirection) < 180) {
            dir -= TURNSPEED;
        } else {
            dir += TURNSPEED;
        }
    }

    sens.setDirection(dir);

}
```

Figure 3. Calculating the best way to turn and turning the bot towards its desired direction.



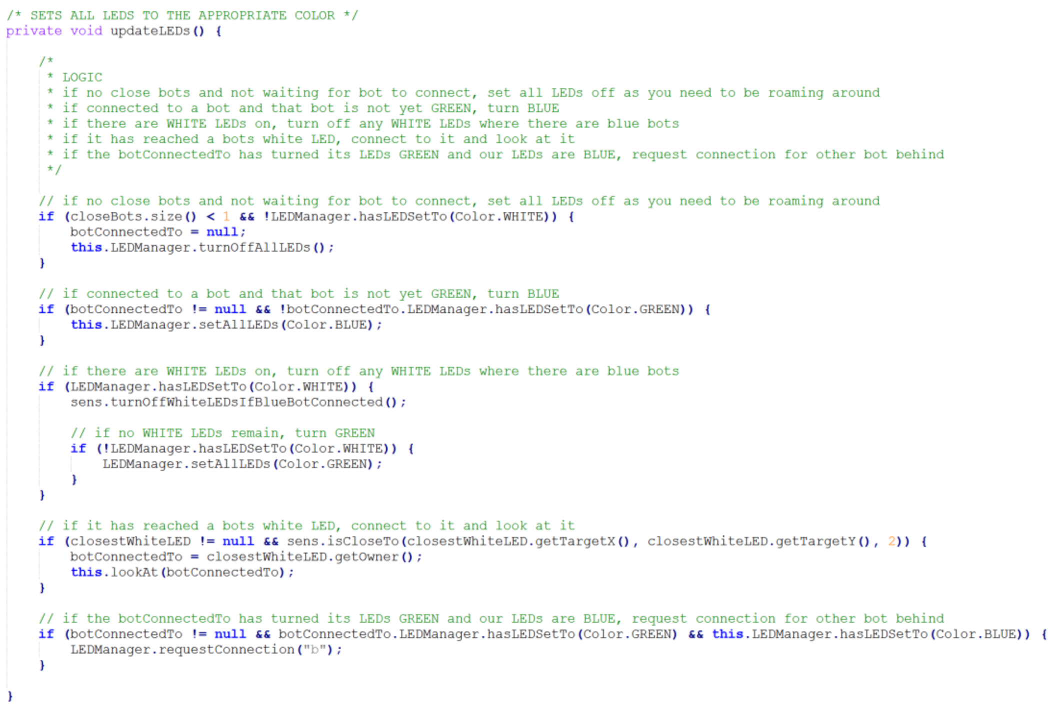
```java
/* MOVES THE BOT FORWARDS IN DIR FACING BY GIVEN AMOUNT */
private void moveForward(double length) {
    sens.setXPos(sens.getXPos() + (Math.cos(Math.toRadians(sens.getDirection())) * length));
    sens.setYPos(sens.getYPos() + (Math.sin(Math.toRadians(sens.getDirection())) * length));
}
```

Figure 4. For the simulator only; calculating a bots' new position after it has moved forward a given distance in the direction it is facing. Updates the bots' new position.



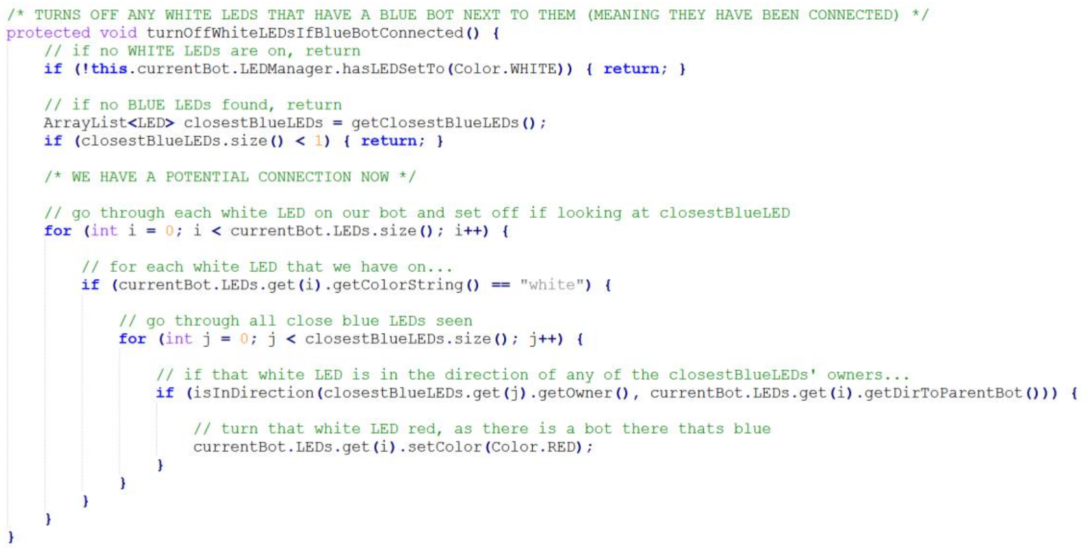
```java
/* SETS THE DIRECTION OF THE BOT TO FACE THE GIVEN X, Y POINT */
private void lookAt(double x, double y) {
    // http://www.gamefromscratch.com/post/2012/11/18/GameDev-math-recipes-Rotating-to-face-a-point.aspx
    desiredDirection = Math.atan2(y - sens.getYPos(), x - sens.getXPos()) * (180 / Math.PI);
    fixDirections();
}
```

Figure 5. Setting the desired direction of a bot to look at a specified position on the Canvas.



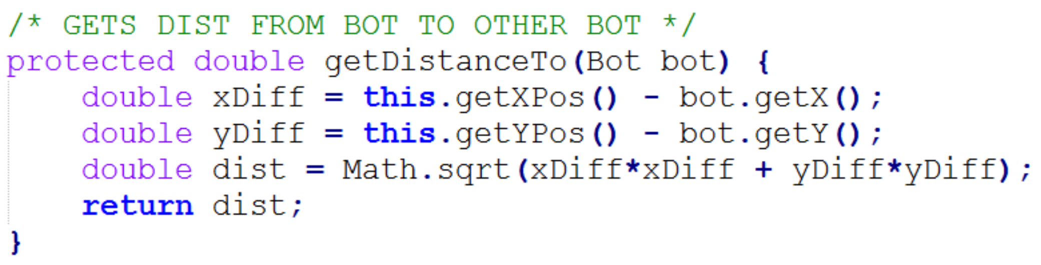
```
/* SETS ALL LEDS TO THE APPROPRIATE COLOR */
private void updateLEDs() {

    /*
     * LOGIC
     * if no close bots and not waiting for bot to connect, set all LEDs off as you need to be roaming around
     * if connected to a bot and that bot is not yet GREEN, turn BLUE
     * if there are WHITE LEDs on, turn off any WHITE LEDs where there are blue bots
     * if it has reached a bots white LED, connect to it and look at it
     * if the botConnectedTo has turned its LEDs GREEN and our LEDs are BLUE, request connection for other bot behind
     */

    // if no close bots and not waiting for bot to connect, set all LEDs off as you need to be roaming around
    if (closeBots.size() < 1 && !LEDManager.hasLEDSetTo(Color.WHITE)) {
        botConnectedTo = null;
        this.LEDManager.turnOffAllLEDs();
    }

    // if connected to a bot and that bot is not yet GREEN, turn BLUE
    if (botConnectedTo != null && !botConnectedTo.LEDManager.hasLEDSetTo(Color.GREEN)) {
        this.LEDManager.setAllLEDs(Color.BLUE);
    }

    // if there are WHITE LEDs on, turn off any WHITE LEDs where there are blue bots
    if (LEDManager.hasLEDSetTo(Color.WHITE)) {
        sens.turnOffWhiteLEDsIfBlueBotConnected();

        // if no WHITE LEDs remain, turn GREEN
        if (!LEDManager.hasLEDSetTo(Color.WHITE)) {
            LEDManager.setAllLEDs(Color.GREEN);
        }
    }

    // if it has reached a bots white LED, connect to it and look at it
    if (closestWhiteLED != null && sens.isCloseTo(closestWhiteLED.getTargetX(), closestWhiteLED.getTargetY(), 2)) {
        botConnectedTo = closestWhiteLED.getOwner();
        this.lookAt(botConnectedTo);
    }

    // if the botConnectedTo has turned its LEDs GREEN and our LEDs are BLUE, request connection for other bot behind
    if (botConnectedTo != null && botConnectedTo.LEDManager.hasLEDSetTo(Color.GREEN) && this.LEDManager.hasLEDSetTo(Color.BLUE)) {
        LEDManager.requestConnection("b");
    }

}
```

Figure 6. Deciding what the colours of each LED needs to be and setting them.



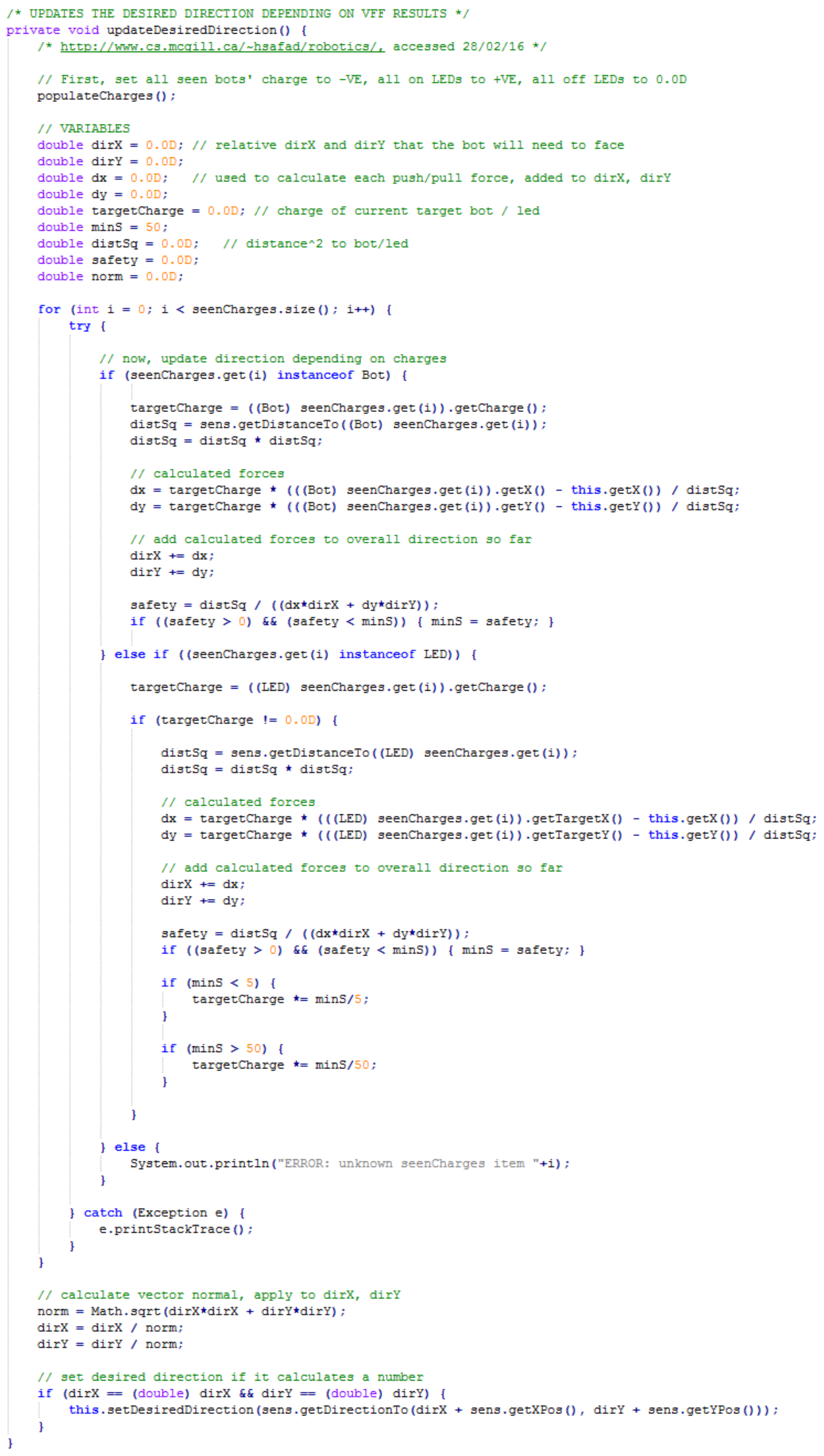
```
/* TURNS OFF ANY WHITE LEDS THAT HAVE A BLUE BOT NEXT TO THEM (MEANING THEY HAVE BEEN CONNECTED) */
protected void turnOffWhiteLEDsIfBlueBotConnected() {
    // if no WHITE LEDs are on, return
    if (!this.currentBot.LEDManager.hasLEDSetTo(Color.WHITE)) { return; }

    // if no BLUE LEDs found, return
    ArrayList<LED> closestBlueLEDs = getClosestBlueLEDs();
    if (closestBlueLEDs.size() < 1) { return; }

    /* WE HAVE A POTENTIAL CONNECTION NOW */

    // go through each white LED on our bot and set off if looking at closestBlueLED
    for (int i = 0; i < currentBot.LEDs.size(); i++) {

        // for each white LED that we have on...
        if (currentBot.LEDs.get(i).getColorString() == "white") {

            // go through all close blue LEDs seen
            for (int j = 0; j < closestBlueLEDs.size(); j++) {

                // if that white LED is in the direction of any of the closestBlueLEDs' owners...
                if (isInDirection(closestBlueLEDs.get(j).getOwner(), currentBot.LEDs.get(i).getDirToParentBot())) {

                    // turn that white LED red, as there is a bot there thats blue
                    currentBot.LEDs.get(i).setColor(Color.RED);
                }
            }
        }
    }
}
```

Figure 7. Deciding whether a bot has connected to any of the LEDs. The method also turns the LEDs off when a bot was found to be connected.



```
/* GETS DIST FROM BOT TO OTHER BOT */
protected double getDistanceTo(Bot bot) {
    double xDiff = this.getXPos() - bot.getX();
    double yDiff = this.getYPos() - bot.getY();
    double dist = Math.sqrt(xDiff*xDiff + yDiff*yDiff);
    return dist;
}
```

Figure 8. Calculating the distance to another bot.



```java
/* UPDATES THE DESIRED DIRECTION DEPENDING ON VFF RESULTS */
private void updateDesiredDirection() {
    /* http://www.cs.mcgill.ca/~hsafad/robotics/, accessed 28/02/16 */

    // First, set all seen bots' charge to -VE, all on LEDs to +VE, all off LEDs to 0.0D
    populateCharges();

    // VARIABLES
    double dirX = 0.0D; // relative dirX and dirY that the bot will need to face
    double dirY = 0.0D;
    double dx = 0.0D;   // used to calculate each push/pull force, added to dirX, dirY
    double dy = 0.0D;
    double targetCharge = 0.0D; // charge of current target bot / led
    double minS = 50;
    double distSq = 0.0D;   // distance^2 to bot/led
    double safety = 0.0D;
    double norm = 0.0D;

    for (int i = 0; i < seenCharges.size(); i++) {
        try {

            // now, update direction depending on charges
            if (seenCharges.get(i) instanceof Bot) {

                targetCharge = ((Bot) seenCharges.get(i)).getCharge();
                distSq = sens.getDistanceTo((Bot) seenCharges.get(i));
                distSq = distSq * distSq;

                // calculated forces
                dx = targetCharge * (((Bot) seenCharges.get(i)).getX() - this.getX()) / distSq;
                dy = targetCharge * (((Bot) seenCharges.get(i)).getY() - this.getY()) / distSq;

                // add calculated forces to overall direction so far
                dirX += dx;
                dirY += dy;

                safety = distSq / ((dx*dirX + dy*dirY));
                if ((safety > 0) && (safety < minS)) { minS = safety; }

            } else if ((seenCharges.get(i) instanceof LED)) {

                targetCharge = ((LED) seenCharges.get(i)).getCharge();

                if (targetCharge != 0.0D) {

                    distSq = sens.getDistanceTo((LED) seenCharges.get(i));
                    distSq = distSq * distSq;

                    // calculated forces
                    dx = targetCharge * (((LED) seenCharges.get(i)).getTargetX() - this.getX()) / distSq;
                    dy = targetCharge * (((LED) seenCharges.get(i)).getTargetY() - this.getY()) / distSq;

                    // add calculated forces to overall direction so far
                    dirX += dx;
                    dirY += dy;

                    safety = distSq / ((dx*dirX + dy*dirY));
                    if ((safety > 0) && (safety < minS)) { minS = safety; }

                    if (minS < 5) {
                        targetCharge *= minS/5;
                    }

                    if (minS > 50) {
                        targetCharge *= minS/50;
                    }

                }

            } else {
                System.out.println("ERROR: unknown seenCharges item "+i);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // calculate vector normal, apply to dirX, dirY
    norm = Math.sqrt(dirX*dirX + dirY*dirY);
    dirX = dirX / norm;
    dirY = dirY / norm;

    // set desired direction if it calculates a number
    if (dirX == (double) dirX && dirY == (double) dirY) {
        this.setDesiredDirection(sens.getDirectionTo(dirX + sens.getXPos(), dirY + sens.getYPos()));
    }
}
```

Figure 9. Calculating the resultant push/pull force from the VFF algorithm.

D. Version List

1.3 (finished 12/04/16)

Multi-target capability has been added, bots can now form various shapes depending on what seed is elected.

The bots are now slightly attracted to the center of the canvas, which removes the need to look at the center when leaving it.

Bug fixes and refactoring to finalise the project.

1.2 (finished 05/04/16)

Bots now gather all data of which bots they're connected to through the LEDs.

Instead of the target system for electing seed bots, the user will select and pick a bot to become a seed.

The button panel has now been removed, as well as any button listeners.

Added instructional text to top left of screen.

1.1 (finished 28/03/16)

When a bot connects to the bot, its LED will switch off and the other bots LED will come on (for now, mimic).

Also, the connecting system has been improved so that the bots will now connect in a straighter line.

1.0 (finished 16/03/16)

LEDs are now controlled by an LEDManager class. This means that the Bot can give high level commands to the LEDs

and not have to micromanage them.

Bots will now connect themselves to other bots properly, and display LED lights confirming so.

A bug has been fixed where the bot can be seen but the LED cannot as too far away.

0.9 (finished 15/03/16)

Bots now have multiple coloured LEDs, only 1 of which has a charge (attraction). The others are used to confirm the bots have connected to another bot.

The user can now place an object in the canvas which will serve as a point/target where a seed robot will elect itself.

This target has an interface and a single unimplemented target type (line).

The positioning system bug has been corrected to make the center of ALL objects its x, y co-ord.

0.8 (finished 03/03/16)

VFF has now been implemented and works properly. The bots can now actively seek out LEDs and move towards them whilst avoiding other bots.

The values for the system has been tweaked.

0.7 (finished 01/03/16)

Various code pieces has been refactored to make the system easier to understand and work more efficiently.

The color scheme has been changed to make the Simulator more aesthetically appealling.

Work has begun on Virtual Force Fields (VFF) to allow the bot to get to a desired position while avoiding other bots.

Although the system is technically implemented, serious 'tweaking' is needed to get it to work properly

0.6 (finished 25/02/16)

LEDs have been increased to 7 per Bot, and the functionality has been improved slightly to make them easier to

use (switch(red);). Bots can now also roam and avoid each-other randomly. This is done by simply going in the opposite

direction to the average vector of close bots.

0.5 (finished 18/02/16)

Bots now have 3 LEDs attached to the class, and these LED’s have basic functions such as switch().

The LEDs also render themselves in the correct part of the screen relative to the bot they're attached to.

0.4 (finished 15/02/16)

Bots now have a desired direction. At every tick, the bot now turns towards the desired direction OR moves forward if it is already facing that desired direction. Bots can now also calculate the average direction relative to itself to all of its close bots.

0.3 (finished 12/02/16)

Users can now interact with the BotCanvas in order to add, move, and remove bots using the mouse.

UI has been cleaned up to be more aesthetically pleasing and the engine has also been 'de-plagiarised'.

0.2 (finished 08/02/16)

This version brings basic bot functionality, and the introduction to the sensor class.

Bots can now roam around the canvas randomly, avoiding the canvas edges, and detect near-by bots.

Bug with the btnPanel overlapping the BotCanvas has been fixed.

Tested with 500 bots at 60 ticks and preformed with 80 - 100 fps.

0.1 (finished 07/02/16)

A basic JFrame with a canvas (previously JPanel) area where bots can be drawn.

No bot functionality has been implemented, only the ability to do basic functions for themselves (moveForward(), turnAround()...)

The simulator uses a plagiarised engine that keeps it running at x ticks.

Basic UI design, buttons, labels, etc.

exit, randomise directions, center all robot buttons have been implemented

Bibliography

[1] Parker, L. (2003*). Current research in multirobot systems*. Artif Life Robotics, 7(1-2), pp.1-5.

[2] Penders, J. (2007). *Robot Swarming Applications.* Materials and Engineering Research Institute, 1-8.

[3] AutoStore. (2016). *AutoStore | Automated Warehouse Robots*. [online] Available at: http://autostoresystem.com/ [Accessed 9 Feb. 2016].

[4] Şahin, E. and Winfield, A. (2008). *Special issue on swarm robotics.* Swarm Intell, 2(2-4), pp.69-72.

[5] Tan, Y. and Zheng, Z. (2013*). Research Advance in Swarm Robotics*. Defence Technology, 9(1), pp.18-39.

[6] Augugliaro, F., Lupashin, S., Hamer, M., Male, C., Hehn, M., Mueller, M., Willmann, J., Gramazio, F., Kohler, M. and D'Andrea, R. (2014). *The Flight Assembled Architecture installation: Cooperative construction with flying machines*. IEEE Control Systems, 34(4), pp.46-64.

[7] *NASA's tumbleweed-inspired rovers for large-scale planetary exploration*. (2008). Industrial Robot, 35(2).

[8] M. Gnatowski, (2006). *Search-and-rescue using team of robots*. Industrial Robot: An International Journal, 9(28).

[9] Penders, J. (2007). *Robot Swarming Applications.* Materials and Engineering Research Institute, 1-8.

[10] O'Grady, R., Groß, R., Christensen, A. and Dorigo, M. (2010). *Self-assembly strategies in a group of autonomous mobile robots.* Autonomous Robots, 28(4), pp.439-455.

[11] D.P. Barnes, P. Summers, A. Shaw, *An investigation into aerobot technologies for planetary exploration*, 6th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2000, pp. 3.6–5

[12] Lis.epfl.ch. (2016). *Laboratory of Intelligent Systems | EPFL*. [online] Available at: http://lis.epfl.ch/ [Accessed 03 Feb. 2016].

[13] Bulletphysics.org. (2016). *Real-Time Physics Simulation*. [online] Available at: http://bulletphysics.org/wordpress/ [Accessed 30 Jan. 2016].

[14] Tuci, E. and Rabérin, A. (2015). *On the design of generalist strategies for swarms of simulated robots engaged in a task-allocation scenario*. Swarm Intell, 9(4), pp.267-290.

[15] Persson, M (aka. Notch). (2016). *Ludum Dare 22 | Ludum Dare*. [online] Available at: http://ludumdare.com/compo/ludum-dare-22/?action=preview&uid=398 [Accessed 13 Feb. 2016].

[16] Mann, S., Mann, S., Nanavati, V. and Silverton, J. (2016). *Mean direction through 360 degrees?* [online] PC Review. Available at: http://www.pcreview.co.uk/threads/mean-direction-through-360-degrees.1746638/ [Accessed 20 Feb. 2016].

[17] GameFromScratch.com. (2012). *GameDev math recipes: Rotating to face a point*. [online] Gamefromscratch.com. Available at: http://www.gamefromscratch.com/post/2012/11/18/GameDev-math-recipes-Rotating-to-face-a-point.aspx [Accessed 28 Feb. 2016].

[18] Phys.org. (2016). *Airborne robot swarms are making complex moves (w/ video)*. [online] Available at: http://phys.org/news/2012-02-airborne-robot-swarms-complex-video.html [Accessed 3 Mar. 2016].

[19] Borenstein, J. and Koren, Y. (1989). *Real-time obstacle avoidance for fast mobile robots*. IEEE Transactions on Systems, Man, and Cybernetics, 19(5), pp.1179-1187.

[20] O'Grady, R., Christensen, A. and Dorigo, M. (2009). SWARMORPH: Multirobot Morphogenesis Using Directional Self-Assembly. *IEEE Trans. Robot.*, 25(3), pp.738-743.

[21] Mulgaonkar, Y., Cross, G., Kumar, V. (2015). *Design of Small, Safe, and Robust Quadrotor Swarms*. IEEE International Conference on Robotics and Automation (ICRA), pp. 2208-2215.

[22] Cs.mcgill.ca. (2016). *Local Path Planning Using Potential Field*. [online] Available at: http://www.cs.mcgill.ca/~hsafad/robotics/ [Accessed 28 Feb. 2016].

[23] G. R. S. Bhattacharya and V. Kumar. (2015). *Persistent Homology for Path Planning in Uncertain Environments*, IEEE Transactions on Robotics (T-RO), vol. 31, iss. 3, pp. 578-590.

[24] Brambilla, M., Ferrante, E., Birattari, M. and Dorigo, M. (2013). *Swarm robotics: a review from the swarm engineering perspective*. Swarm Intell, 7(1), pp.1-41.

[25] Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999). *Swarm intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.

[26] Dudek, Jenkin, Milios. and Wilkes. (1993). *A taxonomy for Swarm Robots*. Intelligent Robots and Systems '93, IROS '93.Proceedings of the 1993 IEEE/RSJ International Conference, 1, pp.441 - 447

[27] Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Retornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stutzle, T., Trianni, V., Tuci, E., Turgut, A. and Vaussard, F. (2013). *Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms*. IEEE Robotics & Automation Magazine, 20(4), pp.60-71.

[28] Koren, Y., and Borenstein, J. (1991). *Potential Field Methods and their Inherent Limitations for Mobile Robot Navigation.* IEEE Conference on Robotics and Automation, 2, pp.1398-1401.

[29] Christensen, A., O'Grady, R. and Dorigo, M. (2007). *Morphology Control in a Multirobot System*. IEEE Robotics & Automation Magazine, 14(99), pp.x5-x5.

[30] Mondada, F., Gambardella, L., Floreano, D., Nolfi, S., Deneubourg, J. and Dorigo, M. (2005). *The cooperation of swarm-bots - Physical interactions in collective robotics*. IEEE Robotics & Automation Magazine, 12(2), pp.21-28.

[31] Ducatelle, F., Di Caro, G., Pinciroli, C. and Gambardella, L. (2011). *Self-organized cooperation between robotic swarms*. Swarm Intell, 5(2), pp.73-96.