

Botond Hamori

Drone Simulator for Information Relaying

B.Sc. Hons Computer Science

24th March 2017

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

The working files and a copy of the report are accessible via this link:
<http://www.lancaster.ac.uk/ug/hamori/FYP.html>

Contents

1	Introduction	6
1.1	Project Description and Overview	6
1.2	General Overview	6
1.3	Requirements – In-Depth Details.....	6
1.3.1	General Requirements	7
1.3.2	The Playground	7
1.3.3	UAS Swarm.....	7
1.3.4	Path	8
1.3.5	Communication System	8
1.3.6	Enemy Units	9
1.4	Summary of Content	9
2	Background	10
2.1	Commercial Applications	10
2.2	Military Applications	10
2.2.1	Currently Available Models.....	11
2.3	Summary	12
3	Design.....	13
3.1	How We Started	13
3.2	Programming Languages.....	14
3.2.1	Analysis of Open-Source Programming Languages	14
3.2.2	Python	14
3.2.3	C programming language	14
3.2.4	Ruby	14
3.2.5	Java.....	14
3.2.6	GUI – What Can Programming Languages Offer?	15
3.3	Description of Project Elements	15
3.3.1	Classes Implemented by My Colleague	15
3.3.2	Classes I Made a Contribution To	15
3.3.3	Classes I Implemented	16
3.4	Description of Algorithms	17
3.5	Summary	18
4	Implementation	19
4.1	Key Elements of Java.....	19
4.2	The Swarm Class	20

4.2.1	Remove drone method	22
4.2.2	Create Packets.....	22
4.2.3	Recreate message	24
4.2.4	Create Drones	24
4.2.5	Move method.....	25
4.3	The Playground Class	28
4.3.1	Swarm Instance	28
4.4	Miscellaneous Classes	31
4.5	Summary	32
5	Development Process	33
5.1	Early Phase	33
5.2	Later Phases	33
5.3	Challenges, Problems and Solutions	34
5.3.1	Movement.....	34
5.3.2	Swarm Shape.....	34
5.3.3	Use of Specific Types of Data	34
5.4	Trial and Error, Testing.....	35
5.5	Summary	35
6	The System in Operation.....	36
6.1	Scenario Description	36
6.2	Starting the Mission and Take Off	36
6.3	Reaching the Transmitter and Message Formatting	37
6.4	Drone elimination	38
6.5	Message Reconstruction and Mission Outcome	39
6.6	Summary	40
7	Process Description.....	41
7.1	What Happens in the Background	41
7.1.1	Drones and Information.....	41
7.1.2	Information Reconstruction.....	41
7.2	Functions and Algorithms in the Code.....	42
7.2.1	Swarm Shape.....	42
7.2.2	Printing the Messages.....	42
7.3	Summary	43
8	Testing and Evaluation	44
8.1	Testing During Development	44
8.1.1	Testing Classes I Implemented.....	44

8.1.2	Testing the Program as a whole.....	45
8.2	Final Tests.....	46
8.3	Evaluation	47
8.4	Summary	48
9	Conclusion.....	49
9.1	Project outcome.....	49
9.1.1	Review of Requirements	49
9.2	Possible improvements.....	50
9.2.1	Program Performance.....	51
9.2.2	Code elegance	51
9.2.3	Extra features.....	51
9.3	Learning outcomes.....	52
9.3.1	Improvement of Technical Skills	52
9.3.2	Improvement of Non-Technical Skills	53
	References	54

1 Introduction

1.1 Project Description and Overview

Drone simulator for information relaying – as the title suggests our aim is to create a simulator which visually represents a battlefield where there is a number of enemy units interacting with a swarm of drones. We will aim to simulate a military mission in which a group of drones start from one base, reach a transmitter, where they pick up some data and they deliver it to a receiver while we also want to ensure the enemies won't be able to gain access to the information if a drone is incapacitated. The challenge is to reconstruct the whole message when the swarm arrives at the receiver.

1.2 General Overview

The main purpose of the project is to deliver an unmanned aircraft system (UAS) simulator. A swarm, made up by several drones, follows a user-defined path to reach a transmitter first (e.g. a remote unit that collects information), then a receiver (e.g., a base). The transmitter will create a message upon the arrival of the swarm, with the packets being numbered from 0 to an upper limit defined by the program. The packets composing the message may carry audio-visual or other types of data. The drones then distribute the received packets among themselves, such that each drone holds a unique combination of packets. There are enemies which can shoot at drones, and if they succeed they can retrieve the data held by that drone – hence why we want to split the data so the enemies won't be able to recreate the whole message using the data carried by only one drone.

The work presented in this report is part of a joint project which entails collaborating with a fellow student, James Caldwell. Details of how we worked on the project and how we divided various tasks between us will be described in further paragraphs.

1.3 Requirements – In-Depth Details

The general objectives of the project are

- To visually represent a battlefield with drone units and enemy military units where units interact with each other
- To simulate a scenario where drones act as information relay systems
- To develop an algorithm which formats the whole message, enabling drones to hold a unique combination of packets and ensuring the enemy won't be able to reveal the whole message
- To develop an algorithm which reconstructs the whole message using received packets

Specific product requirements are detailed in the following sections. Note that different bullet points have been used to indicate requirements related to my part of the project, my partner's part and parts that have been jointly developed. More specifically:

- ✓ Requirements concerning myself
- Requirements concerning James Caldwell
- Requirements concerning both of us

1.3.1 General Requirements

- The simulator **shall** be developed with an open-source software – this was the very first requirement. We had to agree upon the use of an open-source software. Our focal point was experience in different programming languages rather than anything, we chose what we thought would be the greatest option: Java.
- The simulator **shall** use a stand-alone executable file – this will allow the user to input desired values, offering diversity to the simulation.
- The simulator **shall** run on common operating systems – this is very important since we have to make sure the program is compatible with popular operating systems, such as Window, iOS, or even Linux.
- A text file **shall** provide input parameters – there is a number of parameters the user shall be able to define. The format of the file has to be compatible with a word processor. Also, the file must describe the mission.
- The simulator **should** offer a visual interface for input parameters allowing the text file to be modified during a mission – this would make it easier to modify parameters at any time.
- The simulator **should** allow a togglable manual flight mode – this will allow the user to test the capability of the swarm, such as how long the batteries can last, discover an alternative and safer path than the predefined one, or even try to keep the distance from enemies to preserve the drones and securing the message delivery.

1.3.2 The Playground

- The simulator **shall** represent elements in 2D – this will provide the ideal user experience since the user can see everything from the top. It's easier to examine the scene.
- The simulator **should** offer 3D representation of the playground which the user can toggle on and off – this might provide enhanced user experience. The user is able to switch between 3D and 2D representations.
- The playground **shall** have a shape of a rectangle or cuboid and its dimensions shall be given in meters – a core requirement. The simulator depicts a 1000 by 1000-meter area therefore the shape is cuboid.
- The user **should** be able to zoom in or out an area – another requirement which can boost the user experience by allowing the user to examine the scene closer.

1.3.3 UAS Swarm

- ✓ The simulator **shall** display up to 30 UAS units – from a technical point of view it is ideal to cap the maximum number of drones to make the program run smoother. When it comes to user experience this is also a handy feature since the user will want to keep track of the whole swarm and having too many drones is inconvenient.
- ✓ UAS units **shall** be described by speed, power consumption, and energy storage – these characteristics are essential because they provide core functionality for the simulation.
- ✓ Different power consumption for different ranges of speed **should** be defined – another element that can make the simulation more interesting and realistic. When the swarm moves faster it will consume more energy.

- ✓ The swarm **shall** take off from at least one base – a simple yet core requirement for a realistic simulation.
- ✓ The UAS **shall** not take off if the destination on the map is out of range – the user can be warned if they try to reach a point they can't due to insufficient energy.
- ✓ Option to divide the UAS units into swarms **shall** be offered – this provides more freedom for the user, allowing them to create separate swarm and experiment with each one of them.
- ✓ The hierarchy of the swarm **shall** be revised if a UAS goes offline – there should not be any gaps within a circle of drones. If a drone is eliminated from an inner circle, or the leader even, the closest drone in the list will take its position.
- ✓ The input and output arguments of the object describing each UAS **shall** take code modularisation into account – there may be some default parameters provided but nothing must be hard-coded.

1.3.4 Path

- ✓ The path **shall** be defined in 3D – a minor yet another core element. For example, the height of the swarm will affect the UAS's chance of being incapacitated.
- ✓ The trajectory between two points **shall** be a straight line – the swarm has to go from one point straight to the next.
- ✓ A smooth path **should** be generated which goes through identified points – this would greatly improve the visuals since the swarm's behaviour would be more realistic.
- ✓ When manual control is toggled off the swarm **shall** converge to the appropriate path and continue to follow it – the swarm must be able to find its path when manual control is turned off, it should find the closest coordinates and then carry on following the path.

1.3.5 Communication System

- ✓ The communication system **shall** consist of the UAS units, a transmitting unit, and a receiving unit – there are three actors, the transmitter which creates the message, the swarm that acts as a relay, and finally the receiving unit which will try and rebuild the message from the packets delivered by the swarm.
- ✓ The transmitter and receiver **shall** have a fixed position either on the ground or above ground – the swarm has to travel to these fixed locations.
- ✓ Three types of links **shall** be defined: links between the transmitting unit and the swarm, links among UAS units in the swarm, and links between the swarm and the unit – a core element for the communication system which allows the exchange of information.
- ✓ Each link **shall** have a packet erasure probability – this requirement is to make the simulation realistic, there is a chance in real life that a packet becomes absent and the receiving unit will not get the exact same message the transmitting unit sent.
- ✓ The transmitter **shall** broadcast a predefined number of packets when the swarm is within a pre-set range – the swarm has to be close enough in order to pick up packets.
- ✓ Each UAS unit of a swarm **shall** store the successfully recovered packets on its memory – a core element of the communication system.
- ✓ The swarm **should** run an algorithm to redistribute packets among the UAS units so that incapacitated UAS units would not reveal the complete message – ideally we

want to prevent the enemy from obtaining the whole message while ensuring that the swarm carries the complete message.

- ✓ The swarm **shall** relay stored packets to the receiver when it is within range – when the swarm manages to make it to the final point, the receiver, it starts relaying the packets and then the receiver will try to rebuild the message.

1.3.6 Enemy Units

- ✓ The simulator **shall** support up to 10 enemy units – similarly to the number of drones, enemy units need to be tracked by the user. Too many enemies will be distracting and will reduce the chance of mission completion.
- ✓ The enemy units **shall** be on the ground or in the air – there may be different types of enemies.
- ✓ Each enemy unit **shall** follow a path – some might be stationary whereas some might follow a path depending on its type.
- ✓ Each enemy unit **shall** have speed, power consumption, energy storage, firing accuracy, firing rate, and number of rounds – similarly to the UAS units, these are essential for lifelikeness.
- ✓ Firing accuracy **shall** depend on distance – another requirement which makes the simulator more realistic.

1.4 Summary of Content

In this chapter, the main objective of the project was presented, which is to create a UAS simulator program where the drones act as an information relay system. Chapter 2 provides background information on drones, especially on how they are applied. The design of the program is explained in Chapter 3, where possible solutions are discussed for early problems, such as the choice of programming language. Furthermore, the chapter also introduces the project elements, explaining classes and provides a general overview of algorithms. Chapter 4 provides an in-depth, detailed, step-by-step review of the key elements of the code. In Implementation, the algorithms used are also discussed, but unlike in Chapter 3, it is much more technical, since code extracts are provided. The process of development is presented in Chapter 5. The chapter describes early and later phases of the development, challenges James and I faced and possible solutions to these problems. Chapter 6, The System in Operation, provides a scenario and presents the program. This chapter is to provide a feeling of what the program is like to use by detailed description of each event. Process Description, Chapter 7, unveils what happens in the background, explains the outcomes of key functions and algorithms. In Chapter 8, one of the most essential procedures will be described: testing and evaluation. The chapter provides an insight on how we tested the program to meet the requirements. Finally, Chapter 9 brings us to a conclusion which describes the project outcome, possible improvements on the program, extra features that could have been possibly included, and learning outcomes.

2 Background

Unmanned aerial systems, or drones, are used for several purposes, including anything from hobbies, motion picture production and journalism to military applications, research, and aerospace. These devices are versatile, several tools and gadgets can be integrated such as cameras, sensors, or even weapons. Drones come in many shapes and sizes, depending on the purpose of use.

The focus of this project is the military application of drones for relaying information.

2.1 Commercial Applications

There is a number of commercial applications of drones today. Drones are even used in agriculture, they can help farmers to take inventory, study the land, and spray pesticides, fertilizers or water on crops which saves a significant amount of time. ^[16] Drones can also help monitoring animals.

Drones can play a role in architecture as well – they can take pictures of areas where architects wish to construct a building. This helps planning and designing a project. The technique is not only cheap, but it also enhances design accuracy. ^[16]

Environmental monitoring is another area where drones may be applied. The human presence may disturb the ecosystem in certain areas so sending drones that cause less disturbance is a wiser option to monitor the environment. Also, drones can be sent to places that aren't accessible or hard to access by humans. ^[16]

News reporting takes advantage of the use of drones. Aerial recordings can be taken much more easily and cheaper than using other methods. This allows easier media coverage. Drones are also capable of streaming, so real-time update is available for news streaming. ^[16]

2.2 Military Applications

There are two main purposes of using drone in the military; these are **target elimination** and **reconnaissance**. As mentioned, UAS aircrafts can be equipped with weapons, such as missiles. Using drones for such purposes is a common practice today ^[5], however, there have been a few examples of the use of weaponised drones back in the 1980's ^[6]. There is a number of concerns regarding the utilisation of these weapons, therefore there is a list of policies and regulations made by the government – these differ in different countries. The UK, however, does not have a policy for drone strikes ^[3].

The other type of military drones, which we are particularly interested in is reconnaissance aircrafts. These systems are used to explore, observe, and investigate particular areas on the field. For aerial reconnaissance purposes the military sometimes uses miniature UAVs. They are extremely adaptive due to their size; the operator is able to deploy them quickly. Low cost and ease of use are a huge advantage ^[1]. Usually this particular type of drone carry audio-visual type of data. However, due to their relatively small size, it is challenging to get a high quality video because the imaging technology has to be compressed – in other words the system of the camera has to scale with the drone's system in terms of physical size. Thanks to the rate at which technology is developing, there are drones today which provide excellent quality of surveillance – Moore's law is an explanation for this, which states that the number of transistors on a dense integrated circuit double every eighteen months ^[2]. This simply means we can fit more data on a storage device with a specific size.

Besides target elimination and reconnaissance, drones can also be used as a means to extend the communication range of ground-based troops or surveillance equipment. The aim of the project is to create a simulator of a swarm of relay drones, which will collect data from a point on the battlefield and transfer it to another point, for example, a base. The data that each drone carries will compose a message but distribution of the data among nodes will be performed in a way that no single drone has the full message – a technique which is currently unavailable, the project is concerned with this innovation. This allows a more secure relaying across the battlefield. Our aim is to perform a theoretical experiment in which we will try and rebuild the original message at the receiver base using the data carried by the drones. The mission is successful if the receiver is able to rebuild the whole message. This method may be applied in the future as it reduces the chance of compromising data and there is a promising chance for success. For example, if a drone is seized by the enemy, they will be able to gain access to the data stored but will not gain knowledge of the whole message. On the other hand, the base will have the opportunity to receive data from multiple drones and reconstruct the whole message.

2.2.1 Currently Available Models

There are many different models used for both combat and reconnaissance purposes. First, let's look at a particular model utilised in combat, the General Atomics Avenger which bears the nickname "Predator C". This UAS is a recently developed design which specialises in stealth and target elimination. ^[17] Its maximum speed is 745 km/h and can be operated at an altitude of up to 18km. ^[18] The drone is



Figure 2.2.1a: Predator C

equipped with a powerful engine, called PW545 B developed by Pratt & Whitney Canada. ^[18] The engine can develop a thrust of up to 4119lb. ^[18] The system is armed, its arsenal consists of an anti-tank missile, 2 types of guided bombs, a small diameter bomb, and a fuel tank which extends the endurance by 2 hours in case the main resource is consumed during the mission. ^[18] The initial endurance of the UAS is 20 hours. This model is very flexible since it can fulfil different roles: other than combat it may be used for reconnaissance missions, however since the model wasn't particularly designed for exploration, carrying out such mission with this UAS is not optimal since the system suffers from drawbacks such as limited payload carrying capability. ^[18] Only specific reconnaissance missions may be carried out due to the characteristics of the device – size, weight, operation, etc. Both the UK and the USA apply the UAS in their services.

Another specific model we'll look at is called PD-100 Black Hornet, developed by Prox Dynamics (a model is shown in Figure 2.2.1b). This is a miniature UAS device, only weighs 18 grams, including the camera. ^[19] This product is also available for personal use. The endurance of this device is relatively short, compared to the previously mentioned device – it only lasts 25 minutes. Its maximum speed is 5 m/s, significantly slower than the other model. The device is equipped with steerable cameras and live video is enabled. Another key feature

is the autopilot – directions are defined by the operator. ^[19] This UAS can be applied in many situations: search and rescue, reconnaissance, observe inaccessible areas, crowd control, or commercial applications like agriculture. ^[19] The device is inexpensive, safe to use, transportable, and can be easily set on course.

2.3 Summary

In this chapter, the current use of drones was revealed, but our main focus was on the applications of UAS in the army. The project proposes an innovative method for using reconnaissance drones as relays of sensitive information. The project required some time investment into design and planning. In the beginning, there were many possibilities for approaching the project as a problem, decisions were made regarding how to lay down the foundation of the program, including choice of programming language and producing a class diagram.



Figure 2.2.1b: PD-100 Black Hornet, designed for multiple purposes, also available for personal use

3 Design

3.1 How We Started

The most important step when starting a project is to evaluate the requirements. The very first requirement was to use an open-source language for development. There is a list of possible programming languages that could have been used for the project. These languages include Python, C, Ruby, and Java. We decided to choose Java, simply because it is an extremely flexible and dynamic programming language and both me and my colleague had experience using it. It wasn't a hastily made decision to use Java but we both came to a conclusion relatively quickly that this programming language would be the most suitable. More details about our decision to develop the simulator on Java is given in Section 3.2.

Once we've established the language to be used, we moved on to the other requirements. We had to identify specific elements of the project, such as the enemy unit, or the base. We decided to use diagrams to help us visualise the structure of the program. Once we had a clear vision, we decided to create a separate class for each element. My job was to implement the following classes: Swarm, Drone, Receiver, Transmitter. It was clear that we had to work on the Playground class together because that was the class used to represent the objects. Both of us contributed to implementation of the code for the Playground class to represent objects. Figure 3.1 shows a class diagram of the program and explains which classes were developed by me and which classes were jointly developed with my colleague.

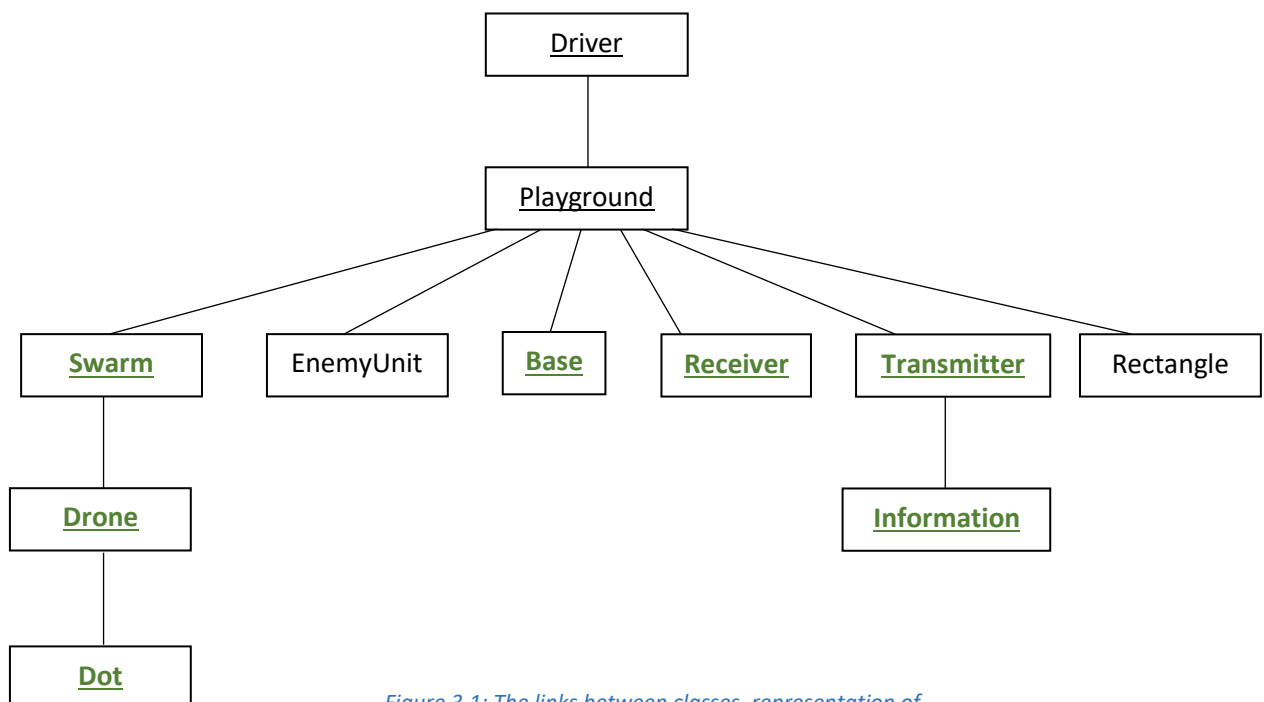


Figure 3.1: The links between classes, representation of relationships.

Green underlined: classes I had to implement

Underlined: classes I contributed to

No formatting: classes I did not need to code at all

3.2 Programming Languages

3.2.1 Analysis of Open-Source Programming Languages

As previously mentioned, a core requirement was to use an open-source programming language. There is a range of suitable programming languages, which will be described and criticised, and justification for deciding to use Java to build the simulator will be provided.

3.2.2 Python

Python is a general-purpose, object-oriented, functional programming language. Key features include code readability, multi-platform, dynamic type checking, extensive standard library, automatic memory management, interactive mode, and GUI programming. Due to the large standard library, it's easy to work with tools already provided. Python also treats everything as an object allowing an easier understanding of object-oriented code ^{[7][8]}.

Since Python is an interpreted language, which means instructions are executed directly, the program may run slow ^[11]. Dynamic type checking can also be an issue since the program may compile and run but won't achieve the desired outcome ^[7].

3.2.3 C programming language

This programming language has been around for a while. It is a static type system which helps preventing unintended or unwanted operations. It first appeared in 1972 and is still in use due to its reliability and efficiency ^[9]. The language provided a foundation for many recently developed languages. A multi-platform language, code written in C requires no change or very little when executing on different operating environments. This language also supports good graphics, meaning the program can have a smooth appearance ^[10].

C doesn't have any runtime checking so debugging becomes challenging. Since C lacks strict type checking, coding has to be done very carefully. For example the language allows to pass a specific type of variable to another one, even though they don't match. Also, there is no support for object-oriented programming. This last factor practically eliminates C from our list of choice ^[10].

3.2.4 Ruby

Ruby supports functional, object-oriented, and imperative programming. Like Python, the language has dynamic type system and automatic memory management. It is also multi-platform, which is essential since we want our program to run on different operating environment. Since neither of us had used Ruby before, it was soon eliminated from the list ^[12].

Ruby truly suffers from substantial issues with processing speed – Java is considered a slow language, but Ruby is about 20 times slower. ^[20] The language is new and unique so it would take some time for one to become confident with using it. The development also causes issues – Ruby relies on a web-based programming platform and new releases aren't compatible. ^[20]

3.2.5 Java

One of the most popular programming languages around the world, Java offers flexibility, object-orientation and simplicity ^[13]. It is a programming language which keeps evolving.

Due to the simplicity of the language, coding and debugging become easy which is vital for complex programs. Another key element is object-orientation. OOP offers a range of features to make it easier to build complex projects ^[14]. For example, when we want to create multiple instances of a drone outside its class, we don't need to recode everything, we can just simply call the method from the class, so if we want to make 20 drones, we may call the method in a for loop. We can use classes to define properties, such as the colour of the drones, their size, or their energy consumption. To meet our requirements, these features are valuable. But other than these, Java also offers flexibility: code written on a Windows machine may run on an iOS or a Linux device since Java is platform independent ^{[7][15]}.

Due to the language's platform independency, Java is slow in comparison to other languages because it runs a virtual machine to run the program. The virtual machine also brings issues because a program may run, but the coding the JVM (Java Virtual Machine) may prevent it from doing so ^[7]. When working on a specific project, we need to import specific packages for each class, otherwise the code will not compile. However, some key methods aren't included in the standard library or not even included at all. Until one of the recent updates, it was impossible directly remove an element from an array and shrink its size.

3.2.6 GUI – What Can Programming Languages Offer?

An outstanding feature Java can offer that the others cannot: user community. This is important, since anyone can create tools and kits that can be used by other Java developers. However, both Python and Java have the same potentials when creating a GUI. ^[21] When it comes to C programming language, the issue is not with the tools or kits used to program a graphical interface, it's the compiler and runtime environment that raises questions. In order to allow the program to display a GUI we are forced to use Visual Studio Express. Despite being an excellent choice for GUI programming, it becomes very complicated to create the desired interface. ^[22] Ruby excels when it comes to the development of web applications, but it loses its usefulness when programming a GUI application. ^[23]

3.3 Description of Project Elements

In this section I will provide a general description of the classes, explaining their purpose and functionality. I will also briefly comment on my role in the development of each class but the actual implementation won't be covered in this section. When it came to a class, we cooperatively worked on and decided to add our own parts of the program. For example, I worked on the Swarm class so it was my job to make it appear on the playground.

3.3.1 Classes Implemented by My Colleague

EnemyUnit: this class provides the enemies. Each enemy can move on a specific path and they are able to shoot at drones when they are within range. The success rate of eliminating a drone depends on the distance. The enemies also have a firing rate and rounds available.

Rectangle: this class is used to paint the background of the playground.

3.3.2 Classes I Made a Contribution To

Playground: this class provides the backbone to the visual representation. Most of the code was implemented by James. My main responsibility was to add the Swarm, Transmitter, and Receiver classes and the Dot class, a class with a less significant role (more details below).

The Playground class is where all the objects are being brought together and displayed, however not all classes and methods are called here.

Driver: similarly to the previous class, my job was to add my own bit of code. Generally, the Driver classes contain the Main method which runs the program. This class is essential since it is responsible for running other classes.

3.3.3 Classes I Implemented

Swarm: the Swarm class, as its title suggests, creates multiple instances of a drone. All functionalities are defined in this class that are responsible for visualisation – methods are called in Playground and there is a range of parameters passed on both from and to the Swarm class. To provide an example, the class is responsible for moving the swarm so there is a method which is called in the Playground. This method passes some parameters (coordinates in this case) to the Playground. When a drone gets eliminated the Playground passes on a parameter acknowledging the Swarm class that there has been a drone removed and the Swarm removes a drone from the group.

Drone: this class contains a few, yet essential variables. To keep track of the drones in the swarm I assigned an ID to each one of them and these identifications are stored in the Drone class. The drones have other characteristics, such as energy level, speed, size, etc. which are all defined in this class.

Dot: the purpose of this class is to enhance the aesthetics of the program. It highlights the next destination the swarm will have to go through on the playground to give the user an idea of where the swarm is heading to.

Base: this class has limited functionality. The appearance is defined in this class, such as colour, shape, and size. The base provides the starting point to the drones.

Receiver: this is very similar to the base, except when the drones are arriving to this location, the receiver unit will try to rebuild the complete message from the data carried by drones. It has no other functionalities, other than defining the appearance.

Transmitter: this is where the swarm picks up the packets. When drones are nearing this location, the transmitter prepares the message and passes it onto the drones. The message is then formatted so that the drones meet the relaying requirements.

Information: the purpose of this class is to create the whole message. It has a function which takes some parameters, one being the length of the message – number of packets – and the other is the number of drones. Using the function with the provided parameters, we can format the message.

3.4 Description of Algorithms

When we started designing our program, we had to take possible algorithms into account. There were two algorithms I decided to implement at some point; both were related to the swarm. The actual implementation of the code will be shown in later sections.

The first algorithm is responsible for maintaining the shape of the swarm – this is not in the requirements, however we found this feature necessary. The shape of the swarm depends on the number of drones available and it updates in real-time. My approach was to get the number of drones and divide this number by the maximum number of drones in a ring within the swarm – the latter value is hard-coded and cannot be changed by the user. If there is no remainder, the program simply draws the whole swarm. However, if there is a remainder, the program draws all rings that can fully be filled with drones and takes the remaining drones to draw the outermost ring. (See Figure 3.4c below).

The second algorithm deals with the communication system. First, the program creates a list of packets and assigns them to the drones. However, to ensure security, the program has to eliminate certain packets from the list before assigning them to drones – we need to make sure incapacitated drones won't reveal full messages. The process of splitting the data involves a formula (or a mathematical expression) which divides the number of drones by the number of packets and multiplies it by a hundred. The result is a rounded integer which defines the number of missing packet from each list. This happens at the transmitter. When the drones get to the receiver, the receiver will try and rebuild the whole information: it takes the list stored by the first drone in the swarm and saves all its content, then it takes the next drone's list, compares the elements of each list and if there is any missing packet the receiver assigns the packet stored by the drone to list of packet stored by the receiver itself. The procedure executes until all drones have been checked. This determines the outcome of the mission, if the whole message has been rebuilt successfully we achieve our goal, otherwise the mission fails.

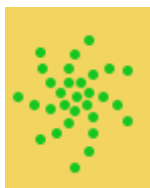


Figure
3.4a



Figure
3.4b



Figure
3.4c

Figures 3.4a, 3.4b, and 3.4c show different formations of the swarm. On Figure 3.4a there are 30 drones in the swarm, this is the largest size of the swarm. On Figure 3.4b there is a swarm with a symmetric form, this is because there are 15 drones – one in the middle and 7 drones in each ring (two rings in total). Figure 3.4c shows an asymmetric formation where the swarm has 28 drones: there are 6 drones in the outer-most ring.

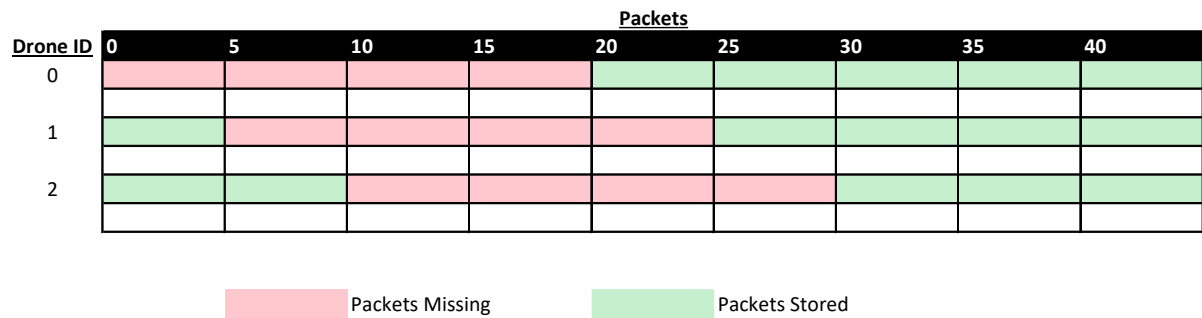


Figure 3.4d: an example of how the first 45 packets would be distributed between the first 3 drones when the swarm has 30 drones. The red lines indicate missing packets; the green ones show the ones held by a specific drone.

3.5 Summary

In this chapter we discovered how we started implementing the program. More importantly, specific program designs were explained, introducing the ideas of each algorithm used, and how the classes were designed to function. We also discovered a number of tools we could possibly use to bring the project into existence. The implementation of the code has not yet been discussed, which is a key element to learn how the program works. Chapter 4, Implementation, will be concerned with the code, explaining core functions and algorithms.

4 Implementation

4.1 Key Elements of Java

Java is an object-oriented language therefore it makes use of **classes**. A class is a basic building block which contains blocks of code. When a class is instantiated by another class, an object is created, allowing to use methods in that class. A class is defined by the keyword `class` and named as `className`, may be done in the following format:

```
public class className
{
    Blocks of code
}
```

Methods can be defined in a class. A method is a collection of statements, and it may or may not take some parameters. This is an extremely useful feature, when a method is declared it can be called at any time, not just from inside the class, but anywhere the class is instantiated. This allows to save time and helps the programmer to keep the code as tidy as possible since there is no need to rewrite the statements, instead just call the method which will execute the block of code.

When creating classes, a **constructor** is either created by the programmer or the program. The constructor must have the same name as its class. This is a special method which is called upon the instantiation of its class. The constructor is always the very first method inside a class. When initialised by Java, it is always empty since it may not be used at all. Here is a pseudo-code representation of the constructor (may follow or be followed by more blocks of code):

```
public class className
{
    public className()
    {
        Blocks of code
    }
}
```

Values and data stored by variables can be acquired or modified using **get** and **set** methods. These are just like any other methods, but named differently, and their names indicate their purpose, for example what specific action they are to perform (acquire or assign/modify), as well as their means to perform the specific action – for example set the value of the variable `a` to 1. The difference between get and set methods is that a get method always returns a value as opposed to a set method, which modifies or assigns a value and it always takes parameters.

```

public int getNumber()
{
    return number;
}

public void setNumber(int a)
{
    this.number = a;
}

```

4.2 The Swarm Class

Inputs

The swarm class has the most variables out of all classes I implemented. ArrayLists were used to store default coordinates and drone ID's. A string array was used to store the label for each point on the map. Double data types were used to define target coordinates and current coordinates because occasionally the calculation won't produce an integer, however these values were converted into integers. The constructor of the class takes a single integer value for each target coordinate, takes an integer to define the layer on the playground, and takes an integer between 1 and 30 to define the amount of drones in the swarm. The leader drone will always be the first drone which has its ID stored at the index 0 in the ArrayList.

Outputs

The swarm class provides outputs for the whereabouts of the swarm. This helps the user to track the swarm by not only acknowledging where the swarm is, but also identifying the next target location. The class also prints the energy left in the drones by displaying the percentage. When the user wishes to input multiple coordinates, they can predict the energy cost of a new mission, based on how long the course would take. When the packets are received from the transmitter and formatted, the user receives a message which confirms the data has been picked up by the swarm. When the drones arrive at the receiver, the system declares the outcome of the mission and prints out the appropriate message to let the user know about the outcome. The program also prints out the remaining energy at the end of the mission.

Important variables and parameters

The class uses a significant amount of variables which are used to calculate values for the movement. There are variables which hold the current x, y, and z coordinates; target coordinates, movement coordinates, and variables to determine the distance between the current position and the next target location. The default coordinates are stored in ArrayLists and they are labelled in a string array. The elements of these ArrayLists are used as parameters for specific functions in the class. The full message is stored and formatted using ArrayLists.

```
int[] listOfXCoordinates = new int[] {120, 190, 260, 330, 400, 500, 600, 670, 740, 810, 880, 950};
int[] listOfYCoordinates = new int[] {730, 710, 490, 470, 250, 50, 250, 470, 490, 710, 730, 950};
int[] listOfZCoordinates = new int[] {25, 30, 35, 35, 35, 35, 35, 35, 35, 30, 25, 20};
String[] points = new String[] {"A", "B", "C", "D", "E", "transmitter", "F", "G", "H", "I", "J", "receiver"};
```

Figure 4.2a: ArrayLists of coordinates and their labels

As shown in Figure 4.1.1a, the coordinates are stored in ArrayLists simply because they are easy to modify. These coordinates define the default path the swarm follows, but the program can take input from the user using a separate input file. Each point is labelled using the letters of the alphabet, however the coordinates of the transmitter and receiver are labelled as “transmitter” and “receiver” respectively. Something else to note is the changes in the z-coordinates. The z-coordinate represents the height of the swarm. When the swarm leaves the base, located at (50, 950) its z-coordinate is 20 and the coordinate gradually increases until it reaches the value of 35 (in units of distance), at point C which is the third point from the base. The transmitter is in the upper section of the map, in the middle – so its coordinates are (500, 50). The receiver opposite to the base in the bottom section of the map, its coordinates are (950, 950) which is the bottom right corner. A graph below shows the path of the swarm. The string array called `points` is printed when the swarm reaches a point, for example “Reached point A, heading to B”.

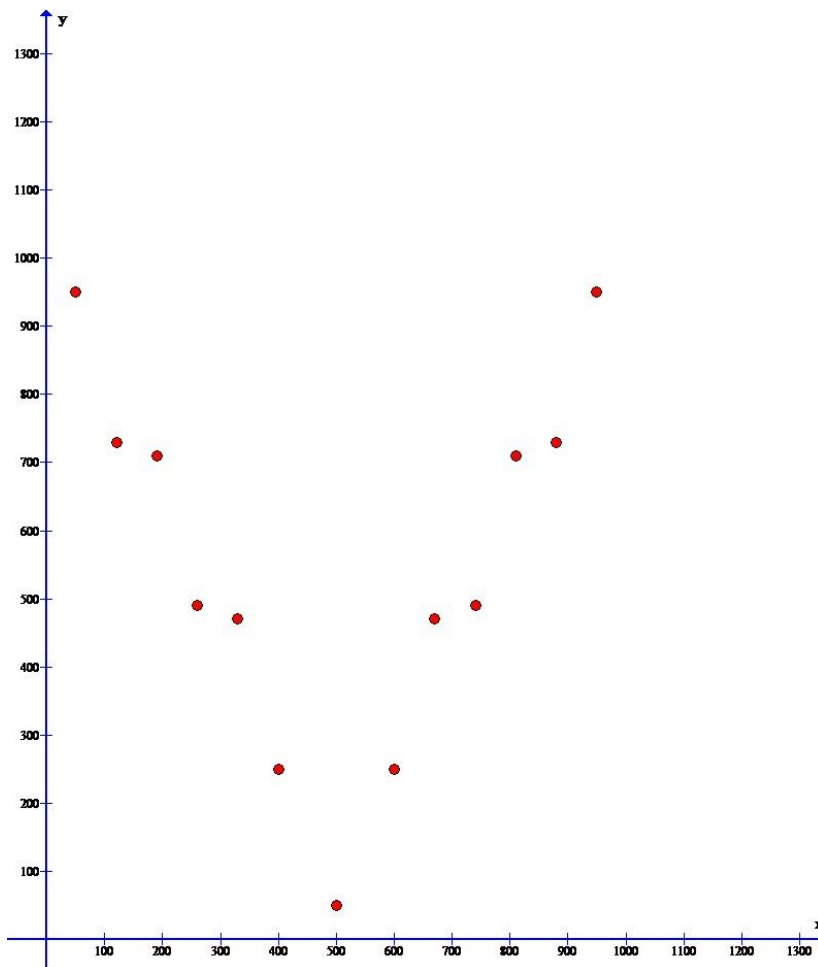


Figure 4.2b: A representation of the default path of the swarm. As you can see the path is symmetric, however it appears upside down when compared to what is displayed by the program, simply because the point (0, 0) in Java is the top left corner. The top left point (50, 950) is the base where the swarm starts its mission, the bottom point (500, 50) is the transmitter, and the top right point (950, 950) is the receiver.

4.2.1 Remove drone method

```
public void removeDrone(int i)
{
    if(i == droneList.get(0))
    {
        System.out.println("Leader's down. New leader: " + droneList.get(1));
    }
    else
    {
        System.out.println("Drone with the ID " + droneList.get(i) + " has been eliminated.");
    }
    droneList.remove(i);
    if(dataStored.size() != 0)
    {
        dataStored.remove(i);
    }
}
```

Figure 4.2.1a: the remove method

When a drone is incapacitated, we want to remove that specific drone from our list. This is achieved by using a set of variables and methods to enable message passing between multiple classes. When the enemy detects a drone, it tries to eliminate it, and if it succeeds the drone is removed. The closest drone's coordinates and its ID is passed on to the `EnemyUnit` via the `Playground` class and, each time the enemy shoots, a Boolean true or false is passed on to the `Playground`, which forwards it to the `Swarm` class. The `Swarm` class checks the value and if it's true, the `removeDrone` method is called. It takes an integer as a parameter which holds the value of the drone's ID. If the drone with the ID 0 is incapacitated, an available drone with the lowest ID becomes the new leader. Also, the data stored by that specific drone is erased, hence the reason we call a remove method on our `dataStored` `ArrayList`.

4.2.2 Create Packets

```
public void createPackets()
{
    System.out.println("Whole message created.");
    System.out.println("Formatting message...");
    fullData = information.getWholeInfo();
    int x = numberOfPackets;
    int y = droneList.size();
    int div = (int) (((double)y / (double)x) * 100);
    int nextIndex = 0;
    for(i = 0; i < droneList.size(); i++)
    {
        ArrayList tempData = new ArrayList();
        for(int j = 0; j < fullData.size(); j++)
        {
            tempData.add(fullData.get(j));
        }
        for(int k = 0; k < div; k++)
        {
            tempData.remove(nextIndex);
        }
        dataStored.add(tempData);
        System.out.println("Drone " + droneList.get(i) + " holds packets: " + tempData);
        nextIndex += (int) (100/y);
    }
}
```

Figure 4.2.2: `createPackets()` method, used to create unique data

When the swarm approaches the transmitter the method `createPackets()` is called. It handles a few `ArrayLists`, their role as follows:

- `fullData`: this is the whole message received by the transmitter. The whole message is stored and formatted.

- `tempData`: this is a temporary ArrayList which is responsible for storing an individual packet, however it will get overwritten as the program executes.
- `dataStored`: this is an ArrayList of ArrayLists which becomes important later, when the swarm delivers the packets. Its role is to hold all information stored by every single drone within the swarm.

The integer `div` is a variable which is used to format the message by subtracting data from the ArrayList at a specific index defined by `nextIndex`.

The method first takes the whole information and then calculates `div` by dividing the number of drones by the number of packets (pieces of information in the ArrayList), then multiplying the result by a hundred. After this the method creates and formats the message for each drone in the swarm. There is two nested for loops, the first loop simply populates the temporary ArrayList by getting the elements of the original ArrayList (`fullData`, our full message) index by index. Then the next loop formats the message, removing a `div` number of data at the index `nextIndex`. The value of `nextIndex` is initialised to 0, and it is incremented by its own value in each iteration. Simply speaking, the leader drone will have packets removed at the index 0, the next drone will have its packets removed at the index `nextIndex`, which is calculated by dividing a hundred by the number of drones. Simple mathematical expressions for finding the indices of the missing packets would be:

$$y_{min} = x * a \text{ (lowest missing index, where } x = \text{nextIndex, } a = \text{drone ID, } y_{min} = \text{index)}$$

$$y_{max} = x * a + b - 1 \text{ (lowest missing index, where } x = \text{nextIndex, } a = \text{drone ID, } b = \text{div } y_{max} = \text{index)}$$

Given we know how many drones we start a mission with and we are able to determine their ID, we can establish the missing packets – anything between y_{min} and y_{max} , including the lower and upper limits. The variable `div` is the number of packets eliminated when formatting the message. When the program has to deal with a higher number of drones, the number of missing packet will be higher when the algorithm assigns the packets to a drone. This is to ensure a fair distribution of packets while also maintaining a similar mission success chance in different scenarios where the number of drones differ.

4.2.3 Recreate message

```
public void recreateInformation()
{
    ArrayList tempList = new ArrayList();
    for(int i = 0; i < droneList.size(); i++)
    {
        tempList = dataStored.get(i);
        for(int j = 0; j < tempList.size(); j++)
        {
            if(receivedData.contains(tempList.get(j)) == false)
            {
                receivedData.add(j, tempList.get(j));
            }
        }
    }

    fullData = information.getWholeInfo();
    if(fullData.equals(receivedData) == true)
    {
        System.out.println("Mission passed. All packets have been delivered.");
    }
    else
    {
        fullData.removeAll(receivedData);
        System.out.println("Mission failed. Missing packets: " + fullData);
    }
}
```

Figure 4.2.3: the method used for recreating the whole message

Upon arriving, the data held by the swarm – ArrayList of ArrayLists – is used to attempt to recreate the whole message. The function `recreateInformation()` is called when the swarm arrives at the receiver. Since an ArrayList containing multiple ArrayLists is used, there is a nested loop in the code. The first loop retrieves the first ArrayList which is the data stored by the drone with the lowest ID. The variable `receivedData` is another ArrayList which stores all elements retrieved from the swarm. Inside the nested loop, there is an if statement. This if statement determines whether the element is already present in the new ArrayList `receivedData`. If the condition is false, the element at the index `j` in `tempList` (a single drone's packet) is added to `receivedData` at the index `j`. This process is repeated until all packets are processed. After processing the information, the received data is compared to the original data, and if they match then we pass the mission.

4.2.4 Create Drones

```
for (id = 1; id < drones; id++)
{
    new Drone(true, energy, 10, id);
    droneList.add(id);
    singleDrone.setDroneID(id);
    System.out.println(singleDrone.getDroneID());

    droneX[id] = this.getDroneX(id);
    droneY[id] = this.getDroneY(id);
    droneZ[id] = this.getDroneZ(id);
}
```

Figure 4.2.4: a loop that creates all drones apart from the leader

The code above only shows what is essential: the loop inside the method. There are a few other variables inside the method which don't appear on the code extract, but these are used to set up the variables used inside the loop and they initialise the values of the first drone – and instance of the drone class is created outside the Swarm method. The variable `id` is a global variable, first used at the first instantiation of the Drone class, assigning 0 as the ID for the first drone, which will become the leader, hence why the loop starts counting at 1. The loop iterates as many times as many drones we need minus one (since one drone has already been created at this point). Inside the loop, we create a new instance of the drone class, assign a Boolean `true` which states that the drone is online; assign energy, size, and ID. Then the drone is added to the list. After the loop is done, the leader drone is established.

4.2.5 Move method

In terms of number of lines, this method is the lengthiest; therefore the code as a whole is not shown in one snapshot. The manipulation of the coordinates was originally implemented by James, I adopted the code and made changes if it was necessary, for example renaming variables and adding a multiplier to speed up the swarm's movement.

```
dot d = new  
  
diffX = this  
diffY = this  
diffZ = this
```

Figure 4.2.5a: the location of the marker for the next target location is set, calculation of the differences between coordinates begins.

These lines are the first few lines within the move method. The location of the `dot` is defined using the get methods of the swarm's next destination on the path. The other parameters set the layer, dot size and dot colour.

The difference between each coordinate is calculated by getting the next destination and subtracting the current position of the swarm. This is important in later parts of the code.

```

if(reached == false) //reached?
{
    if(diffX != 0 || diffY != 0 || diffZ != 0)
    {
        if(diffX < 0)
        {
            xNeg = true;
            diffX = -diffX;
        }
        if(diffY < 0)
        {
            yNeg = true;
            diffY = -diffY;
        }
        if(diffZ < 0)
        {
            zNeg = true;
            diffZ = -diffZ;
        }
    }
}

```

Figure 4.2.5b: test if the swarm has reached the receiver, if not then determine the nature of the numbers.

The Boolean variable `reached` determines whether the swarm has arrived at the final destination of the mission, the receiver (which is usually the base). If the condition is false, the program executes the lines shown in figure 4.1.2.5b.

The next if statement determines whether the swarm has reached the next point. If the difference in x, y, or z are not 0, meaning the swarm is yet to reach the given point, the program manipulates the coordinates. If the differences are negative, the code changes the corresponding Booleans to true (initially set to false) and the values of differences to negative.

```

if(diffX >= diffY && diffX >= diffZ)
{
    moveX = 1;
    moveY = diffY/diffX;
    moveZ = diffZ/diffX;
}
else if(diffX <= diffY && diffY >= diffZ)
{
    moveX = diffX/diffY;
    moveY = 1;
    moveZ = diffZ/diffY;
}
else if(diffX <= diffZ && diffY <= diffZ)
{
    moveX = diffX/diffZ;
    moveY = diffY/diffZ;
    moveZ = 1;
}

```

Figure 4.2.5c: further calculations, set the next coordinates based on the previously determined values

The movement is defined in pixels: the size of the playground is 1000 by 1000 pixels. Using locations of objects, the swarm and the next point on the path in this case, the program calculates in what direction should the swarm move every time the method is called. The first if statement tests the condition between these relations:

```
diffX >= diffY
diffX >= diffZ
```

If the difference between the x coordinates (current position compared to next point) is greater than or equal to the difference between the y coordinates and the difference between the x coordinates is also greater than or equal the difference between the z coordinates, then the swarm moves 1 pixel in the x direction, the value of y is defined as $\text{diffY}/\text{diffX}$ and the value of z is defined as $\text{diffZ}/\text{diffX}$ – these values are always less than 1. Greater difference means the swarm is further away in terms of a single value of a specific coordinate, be it x, y, or z. This if statement concerns the difference of X coordinates.

This block of code finds the highest differences and moves the swarm by 1 pixel on that axis. The movement for rest of the axes is defined as:

$$\text{move}_{\text{axis1}} = \text{difference}_{\text{axis1}} / \text{difference}_{\text{axis2}}$$

Here, axis1 and axis2 represent a specific axis, such as x, y, or z.

```
if(xNeg == true)
{
    moveX = -moveX;
    xNeg = false;
}
if(yNeg == true)
{
    moveY = -moveY;
    yNeg = false;
}
if(zNeg == true)
{
    moveZ = -moveZ;
    zNeg = false;
}
```

*Figure 4.2.5d: assign results
from previous calculations to
new variables*

In order for the swarm to be able to move in a reverse direction the nature of the values has to be determined – whether they are positive or negative. If the Boolean variable corresponding to each coordinate is true, the result from the previous calculation is changed to negative.

```
this.setXPosition(this.getXPosition() + 5*moveX);
this.setYPosition(this.getYPosition() + 5*moveY);
this.setZPosition(this.getZPosition() + moveZ);

this.setEnergy(energy - 1);
singleDrone.setEnergyLevel(energy - 1);
percent = (singleDrone.getEnergyLevel()*100)/1500;
```

*Figure 4.2.5e: apply
results*

The results of the coordinate manipulation are applied here. Using set methods from this class, the result is multiplied by a value (depending on how fast the swarm needs to be) and added to the current coordinate, which is obtained using a get method.

One unit of distance made the swarm reduces the energy by one and the percentage is calculated by retrieving the energy level, multiply it by 100 and divide it by the maximum energy. The energy remaining is calculated whenever the `move()` method is called, so it constantly refreshes.

```

else if (j < points.length - 1)
{
    j = j + 1;
    this.setPathX(listOfXCoordinates[j]);
    this.setPathY(listOfYCoordinates[j]);
    this.setPathZ(listOfZCoordinates[j]);
    new dot(this.getPathX(), this.getPathY(), 15, 7, "BLUE");
    System.out.println("Heading to " + points[j] + ", remaining energy:" + percent + "%");

    if(this.getXPosition() == 500 && this.getYPosition() == 50)
    {
        System.out.println("Reached the transmitter.");
        information.update();
        System.out.println("Packets have been picked up.");
        this.createPackages();
    }
}

```

Figure 4.2.5f: track the whereabouts of the swarm, print appropriate outputs

When the swarm reaches a point, meaning all differences are equal to 0, the program sets the next point in the path. The first line in this else-if statement increments the value of `j` by 1, which is used as an index to retrieve an element from the `ArrayList`. The coordinates of the next point are then set to the values obtained from the `ArrayList` and a new instance of the `dot` class is created, relocating it to the next point on the path. The if statement inside this else-if checks whether the swarm has reached the transmitter. The coordinates are hard-coded since the position of the base, transmitter, and receiver are fixed. When the swarm arrives at the transmitter, an `information.update()` method is called which is a method in another class that creates the whole message. Finally, the packets are formatted and assigned by calling the `createPackages()` method.

4.3 The Playground Class

This class contains a significant number of variables, methods, and functions therefore the focus will only be on parts I implemented. A few variables are listed for each instance; a brief explanation of their purpose is provided.

4.3.1 Swarm Instance

Inputs

The `Playground` class takes some inputs from the swarm. The program first creates an instance of the `Swarm` and the `Dot` class. Using get methods, the `Playground` class obtains the necessary parameters to determine the coordinates of the leader drone, the number of drones to be drawn, a single drone's size and colour, and the layer. The position of the `Dot` depends on the next target location's coordinates.

Important variables

Most of the obtained values are assigned to variables. The code dedicates some variables to store the value of each coordinate – the class is only concerned about the centre location of the swarm this is because multiple instances of `Drone` is created in the `Swarm` class and the `Playground` determines their positions using appropriate set methods. The swarm has a unique formation, depending on how many drones are available. In order to create a unique

formation, mathematical methods are used. The Playground has variables to store specific values used for the mathematical operations.

```
if(o instanceof Swarm)
{
    Swarm a = (Swarm) o;

    double dotX = a.getPathX();
    double dotY = a.getPathY();
    g.setColor(this.getColourFromString(a.getDotColour()));
    g.fillOval((int) (a.getPathX()-a.getDotSize()/2), (int) (a.getPathY()-a.getDotSize()/2), (int) a.getDotSize(), (int) a.getDotSize());

    g.setColor(this.getColourFromString(a.getColour()));
    g.fillOval((int) (a.getXPosition()-a.getSize()/2), (int) (a.getYPosition()-a.getSize()/2), (int) a.getSize(), (int) a.getSize());
}
```

Figure 4.3.1a: create instances of the class Dot and Swarm

This is the first few lines of the block of code, which draws the swarm on the playground. An instance of the Swarm is created; the location of the next point on the path is defined and drawn.

The program draws the object using the `g.fillOval()` method. It takes some parameters; the code above shows the 2D representation since the method takes parameters for the x and y coordinates only. The first two parameters are the coordinates, the third and fourth parameters are the size. All the values are cast to an integer data type since the program uses pixels as coordinates and these have to be whole numbers.

```
int drones = a.getArraySize()-1;
double centreX = a.getXPosition();
double centreY = a.getYPosition();
double centreZ = a.getZPosition();
a.setDroneX(centreX, 0);
a.setDroneY(centreY, 0);
a.setDroneZ(centreZ, 0);
double diameter = a.getSize();
double radians;

int dronesInRing = 7;

float fraction = drones/dronesInRing;
int numberOfRings = (int) Math.round(fraction);
int rem = drones%dronesInRing;

int count = 0;
```

Figure 4.3.1b: variables

The variables used to draw the swarm are shown in figure 4.1.2.1b. The location of each drone isn't defined in the swarm class, instead their position is in relation with the centre of the swarm. Hence why the playground gets the coordinates from the Swarm class and uses it as a centre position. Other variables such as diameter, radians, fraction, and `numberOfRings` are used to calculate the position of a drone.

```

if(rem == 0)
{
    while(count != drones)
    {
        for(int count2 = 1; count2 <= numberOfRings; count2++)
        {
            for(int count3 = 1; count3 <= dronesInRing; count3++)
            {
                count++;

                radians = 2*Math.PI*count3/(dronesInRing) + (Math.PI/2)*(count2-1);

                int newX = (int)(centreX + 7*count2*Math.sin(radians) - diameter);
                int newY = (int)(centreY + 7*count2*Math.cos(radians) - diameter);
                g.fillOval(newX + (int)a.getSize()/2, newY + (int)a.getSize()/2, (int)a.getSize(), (int)a.getSize());

                a.setDroneX(newX, count);
                a.setDroneY(newY, count);
            }
        }
    }
}

```

Figure 4.3.1c: the code which executes when there is no remainder

If there is no remainder that means the swarm can perfectly be displayed with 7 drones in each ring and a drone in the middle. There is a while loop which checks if all the drones are drawn. The first for loop counts the number of rings in the swarm, the nested for loop counts the number drones inside a ring. The inner for loop iterates until all drones in the ring have been displayed, also incrementing the value of count by 1 in each iteration. A mathematical equation can be formed to represent the location of a drone:

$$r = (2\pi * x) / y + (\pi / 2) * (z - 1)$$

Let r = radians, x = count3, y = drones in ring, z = count2. The result of the equation is then applied to define the coordinates. Another mathematical equation for this would be:

$$x = x_{\text{centre}} + 7 * z * \sin(r) - d$$

$$y = y_{\text{centre}} + 7 * z * \cos(r) - d$$

Let d = diameter, x = x coordinate, y = y coordinate. Variables z and r are used in the previous equation; therefore, they stand for count2 and radians respectively.

```

else
{
    while(count != (drones-rem))
    {
        for(int count2 = 1; count2 <= numberOfRings; count2++)
        {
            for(int count3 = 1; count3 <= dronesInRing; count3++)
            {
                count++;

                radians = 2*Math.PI*count3/(dronesInRing) + (Math.PI/2)*(count2-1);

                int newX = (int) (centreX + 7*count2*Math.sin(radians) - diameter);
                int newY = (int) (centreY + 7*count2*Math.cos(radians) - diameter);
                g.fillOval(newX + (int)a.getSize()/2, newY + (int)a.getSize()/2, (int)a.getSize(), (int)a.getSize());

                a.setDroneX(newX, count);
                a.setDroneY(newY, count);
            }
        }
    }
    for(int count3 = 1; count3 <= rem; count3++)
    {
        count++;

        radians = 2*Math.PI*count3/(rem) + (Math.PI/2)*(numberOfRings);

        int newX = (int) (centreX + 7*(numberOfRings + 1)*Math.sin(radians) - diameter);
        int newY = (int) (centreY + 7*(numberOfRings + 1)*Math.cos(radians) - diameter);
        g.fillOval(newX + (int)a.getSize()/2, newY + (int)a.getSize()/2, (int)a.getSize(), (int)a.getSize());

        a.setDroneX(newX, count);
        a.setDroneY(newY, count);
    }
}

```

Figure 4.3.1d: in most cases, there is a remainder and this code is executed

This block of code is slightly different to the previous one. When the program deals with a remainder, this will execute. The first while loop iterates until the counter reaches the value of (number of drones - remainder) which will always give a value that is divisible by 7. The first loop creates the inner rings of the swarm that can fully be filled. The next loop, outside the while loop, a for loop creates the outer most ring which will contain all the remainder drones, always less than 7 if this block of code executes.

Instance of Dot

An instance for the `Dot` is also created, the coordinates however are defined when `Swarm` is being instantiated. The `dot` is being drawn when the instance is created, not when its parameters are defined in `Swarm`.

Instances of Base, Transmitter, and Receiver

Individual instances are created for each of the bases. The base and the receiver appear in the bottom left and the bottom right corner respectively, the transmitter appears on the top of the screen in the middle. For aesthetical purposes, the transmitter appears to be smaller than the base or the receiver.

4.4 Miscellaneous Classes

Classes described in this section have a minor role in the program.

Base: This class defines the appearance of the base. Its x, y and z coordinates, size, and colour are defined in another class, but the constructor takes parameters for each of these variables so the base can be displayed when the instance is created.

Dot: None of the variables in the class is hardcoded, they all depend on values of parameters passed on by the Swarm class. The constructor takes 5 parameters: x and y coordinates, layer, size, and colour. The class also has a few getters and setters.

Information: This class only acts when the swarm reaches the transmitter. An instance of Information is created in the Swarm class and a method is called to create the whole message. The message is stored in an ArrayList which is then passed on to the Swarm class. The constructor takes one parameter, an integer which defines the number of data packets in the whole message.

The drone class: Every time an instance of a drone is created this class is instantiated in the Swarm class. Parameters are passed from the Swarm class to define a single drone in the Drone class. The constructor takes 4 parameters: a Boolean true or false to determine whether the drone is online, an integer to specify the amount of energy a drone has, an integer to define the speed, and another integer to assign an ID for the single drone created.

The transmitter class: The transmitter class has a constructor, which takes 5 parameters: the three coordinates (x, y, and z), an integer to define its diameter, and a string to define its colour.

The receiver class: This class is similar to the base and transmitter class, it takes the same parameters and when this class is instantiated it makes the receiver appear on the screen based on the parameters provided.

4.5 Summary

In Implementation, the core concept behind the code was revealed, explaining the key functions, methods, and algorithms implemented. Some of the key features of Java, such as classes and methods, were also introduced. Both the Swarm and Playground classes contain algorithms which required the implementation of mathematical formulas, another part of the code which this chapter described. How we developed the code both individually and cooperatively is to be covered. In Chapter 5, we will discover the key decisions made both individually and as a team, either when it came to solving a problem or proposing new ideas for the development of the program.

5 Development Process

This chapter describes how each of the classes were developed. Every key step I made and steps we made as a team is outlined in the following sections. The chapter focuses on both early and later phases, as well as presenting the challenges I encountered and other problem we needed to deal with as a team. The functionality and purpose of each class are described in Chapter 4.

5.1 Early Phase

The first step we made was to read and study the requirements to ensure our objectives are clear. Once understanding the project, we discussed and established what programming language to use. We mutually agreed to use Java programming language since both James and myself found this language the best option because we both had enough experience using it.

After making the decision regarding the programming language, we started to propose ideas for building the program. Using class diagrams, at this point our task was to identify the most important classes, these were the Driver, Swarm, Drone, EnemyUnit and Playground. Following the establishment of the program structure we decided to work on separate classes by dividing the work.

When the programming started, our aim was to build a basic program which could then be developed further. To share code, we decided to use Dropbox. There, our shared code was accessible at any time which allowed us to develop our own code easily. We also agreed not to upload malfunctioning code since it would likely to affect the behaviour of other classes due to their strong relationships.

5.2 Later Phases

We kept each other updated over social media to inform each other when significant changes were made to the code. Also, this allowed us to briefly discuss issues in a short period of time. Weekly meetings were arranged as well to discuss problems and propose ideas. This was crucial because we co-developed the code which requires good communication. The reason why good communication was key is because it promotes good understanding of ideas, problems, and possible solutions. Regular verbal updates were also necessary due to task dependencies within the code, for example an enemy wouldn't be able to fire at a drone when the swarm did not provide the drone's coordinates, even if there was a method implemented for this purpose in the enemy's code.

As the code evolved, it became increasingly more complex and harder to read. We had to understand all the code, not just our own because we had to know how functions and methods work in all classes, so when we implemented new functions, we would know the required parameters, inputs, and outputs. For enabling the enemy to fire at a UAS I had to implement set and get methods which allow the enemy unit to acquire the coordinates of the drones in the swarm.

5.3 Challenges, Problems and Solutions

5.3.1 Movement

One of the toughest problems I encountered is to make the movement of the swarm free from bugs. Specifically, the calculations gave unexpected results, which forced the swarm to go in unintended directions. My approach brought me very close to the solution, eventually James had a working code which I adopted. When comparing my code to James' implementation, I realised the issue was with the handling of negative numbers. This caused the swarm to go in opposite directions.

5.3.2 Swarm Shape

I desired to create the swarm a special shape, depending on how many drones are available. When I first created the swarm, it had a perfect shape of a star. I decided to use geometry to define the coordinates of each drone on a circle. However, the result (star shape) was not intended. Therefore, I had to rotate each ring of drones around the centre in order make appear as intended. Initially I worked with a basic mathematical equation which distributes the drones evenly in a circle. The formula had to be improved, using the equation from a previous chapter, the initial mathematical expression for each of the coordinates was the following (the formula is not complete):

$$r = (2\pi * x)/y$$

$$x = x_{\text{centre}} + 7 * z * \sin(r) - d$$

$$y = y_{\text{centre}} + 7 * z * \cos(r) - d$$

Here, the problem was that the value of r would always remain the same, meaning the angle between a drone on the circle and the drone in the middle would remain the same. Hence the reason why the number of rings had to be considered – as the program progresses with drawing the swarm, the counter for the currently treated circle increases. Each time a new circle is drawn by the program, the drones are shifted 90 degrees, which is achieved by multiplying this value in radians by the given counter. This prevents all drones from being in a perfectly straight line which results in forming a star-shaped swarm.

5.3.3 Use of Specific Types of Data

The use of ArrayLists became challenging when the communication system was implemented. Initially, there were variables for storing the full data provided by another class, a variable for temporarily storing the whole list, and another variable for storing all formatted messages. The ArrayLists were declared as global variables which caused the issue. The problem occurred when the program tried to recreate the whole information using information stored on the memory of each drone. The idea of the approach is to acquire the whole information, store it in a temporary variable for formatting purposes, and once done formatting, the program stores the result in an ArrayList of ArrayLists then clears the temporary variable. Then the program takes the whole information, stores it in the temporary variable again, and the cycle goes on until all drones have some packets assigned. However, the elements of the ArrayList was impossible to access as the data was either erased, overwritten, or corrupted by the program. The solution to this problem was to declare the

temporary ArrayList as a local variable so it can only be accessed by the method that uses it. The global variable used to store all ArrayLists remained a global variable.

5.4 Trial and Error, Testing

Since the program was co-developed, we had to ensure our codes were compatible and wouldn't cause any malfunctions. Each time before sharing a newly developed code – meaning overwriting previous working versions – we had to ensure it was working as intended. If there were any unexpected outcomes, the code was revised. Sometimes it took trial and error to deal with such problems, for example the previously mentioned problem with the swarm shape, although it didn't affect the program's performance, it was merely an aesthetic feature. There were several other unexpected outcomes, such as the rate at which drones were incapacitated – a problem regarding a class developed by James. Another example is when the data was formatted in such way that the receiver was unable to reconstruct the whole message, regardless of the circumstances. The issue was that the drones held information which was composed of insufficient amount packets therefore the mathematical equation was changed to achieve an optimal result.

5.5 Summary

In this chapter, the whole process of developing the program was described. Starting with the early phases of the process, it was explained how and why we made certain decisions. Once establishing the appropriate programming language, we decided to visualise the program structure, using diagrams, which helped us to split up the work. As we advanced with the progress, we decided to arrange weekly meetings to keep each other updated.

This chapter also reflects on specific problems encountered during the development process by myself and problems we faced as a team. Identifying the problem, possible approaches and final solutions are also explained. We had to meet our own expectations as well as the project requirements, we wanted to ensure the program has a user friendly GUI, and it is also satisfying to use. Chapter 6, The System in Operation, is to describe what it feels like to use the program.

6 The System in Operation

6.1 Scenario Description

Let's suppose we intend to start the mission with a swarm of 30 drones which follow the default path defined by the program. These variables, the number of drones and coordinates of points on the path, may be defined by the user. Our aim is to use the drones as a relay system to deliver packets from the transmitter to the receiver. When we have a full-size swarm (30 drones) upon the arrival at the transmitter that means x number of packets will be missing from each drone's storage. A mathematical formula to determine the value of x would be the following:

$$x = (a/b)*100 \text{ (where } a = \text{number of drones, } b = \text{number of packets)}$$

This way the message is formatted and delivered to the receiver. If the receiver can successfully perform a reconstruction of the received data, meaning the whole message is rebuilt, then the mission is successful. Otherwise, when all drones have been incapacitated or the number of drones remaining is insufficient, the mission fails.

6.2 Starting the Mission and Take Off

When the mission starts a swarm of drones is launched from a base. The size of the swarm can be defined but the program allows the swarm to have a size of up to 30 drones. The base has the location (50, 950) which is in the bottom left corner. The next point on the path is immediately shown which is represented by the blue dot.

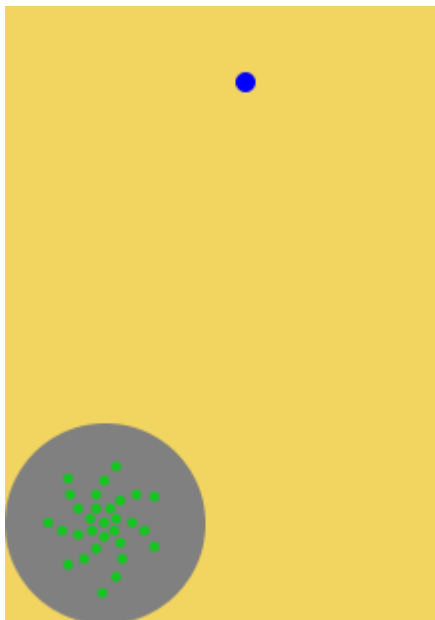


Figure 6.2a: The mission starts with 30 drones that take off from the base in the bottom left corner. The next target location is marked by the blue spot

The system also prints messages to keep the user updated about the progress of the mission. The ID of each drone created is printed and the user is informed when the mission starts. The user is also informed about what the program is expected to do: tracking energy level and displaying destinations using their labels. The next target location is also announced, which is "location A".

```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
The drones are all set, swarm is created. Now tracking energy level and destination points.
Heading to A.

```

Figure 6.2b: The message printed in the console informing the user about the progress of the mission.

6.3 Reaching the Transmitter and Message Formatting

The swarm goes through multiple points on the path before reaching the transmitter. In real life these changes in the path may indicate safe passages, obstacles the swarm attempts to avoid, or other environmental factors.

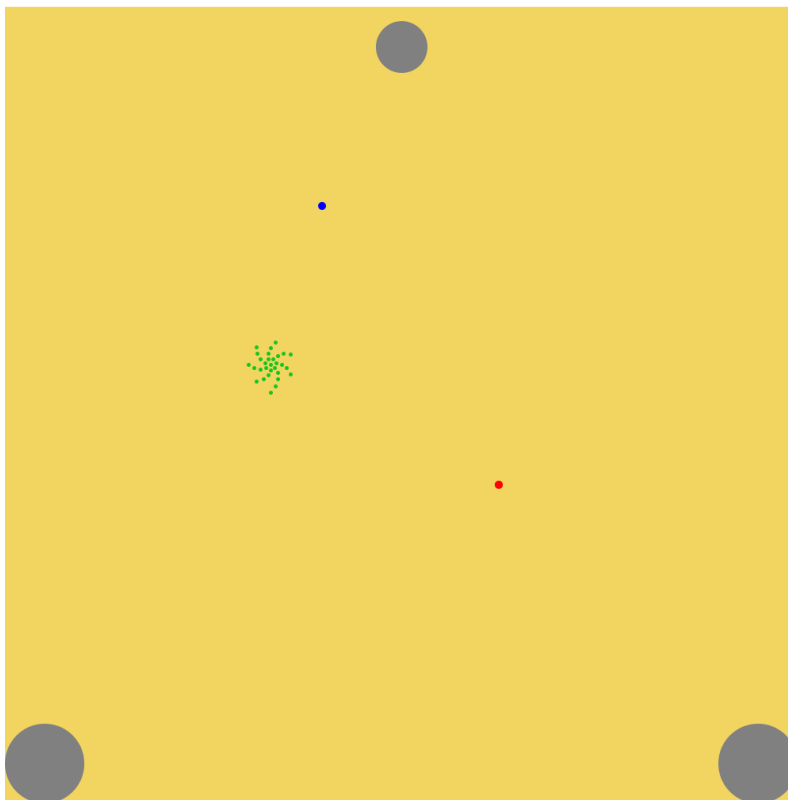


Figure 6.3a: the swarm is en route to the transmitter. There are two enemies (only one enemy is visible, their paths intersect) in this scenario, may be changed by the user.

The energy level is not displayed until reaching the first location because the program only prints the energy level when reaching a target location. The swarm has 5 target locations between the base and the transmitter (excluding the transmitter itself), and another 5 points between the transmitter and the receiver – again, this excludes the receiver as a target location. In other words, there are 10 points in total on the battleground the swarm has to go through which are zones where drones may be incapacitated by enemies.

```
Heading to A.
Heading to B, remaining energy:77%
Heading to C, remaining energy:76%
Heading to D, remaining energy:73%
Heading to E, remaining energy:72%
```

Figure 6.3b: message is printed to allow the user to see which points the drones are heading to and how much energy they have.

Upon reaching the transmitter, the whole message is created, formatted, and a unique combination of packets is stored on each drone's memory. The progress of creating the message is shown on the console as a message output.

```
Whole message created.
Formatting message...
Drone 0 holds packets: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 21
```

The program also provides information on which drone has been incapacitated. However, the lost packets remain a secret so the user won't be able to reconstruct the whole message manually in case the mission fails.

```
Drone with the ID 21 has been eliminated.
Heading to I, remaining energy:60%
Drone with the ID 20 has been eliminated.
```

Figure 6.4b: the message output provided by the program to inform the user which drone has been incapacitated.

6.5 Message Reconstruction and Mission Outcome

The program determines whether the mission succeeded, and based on the outcome it provides a message. In two cases a mission may fail: all drones are eliminated by the enemy units, or the number of drones in the swarm is insufficient therefore a failure in the reconstruction of the whole message is inevitable.

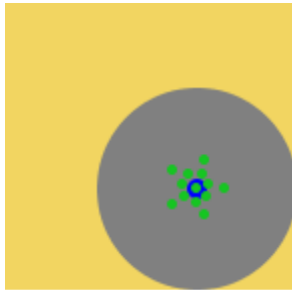


Figure 6.5a: the drones arrive at the receiver. The coordinates are (950, 950) and when the leader reaches this point, message reconstruction begins. Based on the result the outcome is determined and printed.

When the mission ends, the program prints a number of messages to acknowledge the user about the outcome. The message depends on the nature of the mission's outcome: whether it was a success or a failure, and if it failed, what caused the result. When the swarm successfully arrives at the receiver, the program prints out the reconstructed message, the outcome of the mission, the number of remaining drones, and the remaining energy. This allows the user to determine how efficient and effective their attempt is to complete the mission, based on the provided input parameters.

```
Reconstructed information: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 8
3, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 13
3, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149]
Mission passed. All packets have been delivered.
Remaining drones: 13
Remaining energy:53%
```

Figure 6.5b: the whole message printed by the program at the end of the mission. In this case the mission succeeded.

The efficiency and effectiveness of carrying out a mission depends on the user. The program allows to simulate different environments; therefore, the user is invited to input their parameters. When simulating different environments, the user must be aware of certain areas the swarm should not go, for example there is a building which prevents the swarm from going from one point to another in a straight line, instead the drones must fly around the building. To make the mission more complicated, on one side of the building there may be a number of enemies located, but opting to fly the other direction will significantly increase the

duration of the mission – so the user will have to decide to take the shorter or safer path. If the user provides the accurate coordinates the program will simulate and help to determine the optimal path to take by providing the possible outcome of the mission.

6.6 Summary

This chapter presents the program and its aim is to give a feeling of what the system is like to use. A basic scenario was provided in which the swarm consists of 30 drones with 150 packets in the full message and it follows the default, pre-defined path with two enemies trying to eliminate drones. All the results are shown and narrated. However, what exactly happens behind the scenes is not yet described – Chapter 7 will focus on this, providing a more technical overview of the events taking place.

7 Process Description

7.1 What Happens in the Background

This chapter describes how key events, such as the formatting of the message to be relayed by the drones, take place. In the description, snippets of code and screenshots are provided.

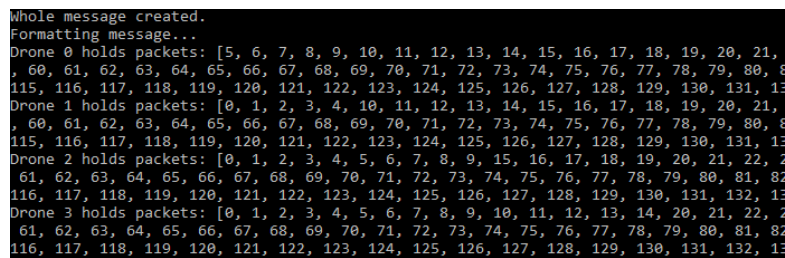
7.1.1 Drones and Information

When the leader of the swarm reaches a specific point, the transmitter creates the full message, formats it, and then assigns it to the drones. This makes use of two ArrayLists: one of them being the list of drones, the other one is the list of data the whole swarm carries.

```
public ArrayList<Integer> droneList = new ArrayList<Integer>();  
ArrayList<ArrayList<Integer>> dataStored = new ArrayList<ArrayList<Integer>>();
```

The reason for using two ArrayLists is when a drone becomes incapacitated, it is eliminated from the list at its index. The corresponding element is also removed from dataStored at the same index. This keeps the two ArrayLists synchronised.

The full message is hidden from the user when the drones pick up the packets, although the data stored is printed.



```
Whole message created.  
Formatting message...  
Drone 0 holds packets: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 
```

only new elements by determining whether the scanned element from the new list is already present in the list that is used to store the received packets.

7.2 Functions and Algorithms in the Code

This section is concerned about specific algorithms that influence the events that take place during the course of the mission.

7.2.1 Swarm Shape

The shape of the swarm always depends on the number of drones. This is updated in every frame in the Playground class. When a drone is incapacitated, the size of the array shrinks. A function in the swarm class passes the value of the size of the array to the Playground class, this way the method implemented in Playground is able to determine how many drones are present. When the enemy eliminates a drone, it sends a message to the Swarm class via Playground and using this message the swarm is updated.

```
if( tempEnemy.getHit() == true )
{
    tempEnemy.setHit(false);
    tempSwarm.setShot(true);
    tempSwarm.removeDrone(k);
}
```

A method called `trackObjects()` in Playground allows the enemy unit to identify a drone. If the enemy has a line of sight it starts firing and if it successfully hits the drone, it passes a Boolean true to this method as well as passing the ID (value of `k`) of the incapacitated drone. The Swarm class is instantiated inside this method, so in this if statement we are able to call methods from Swarm, such as the remove drone method. This way the enemy is able to incapacitate drones.

7.2.2 Printing the Messages

The user is kept updated on the swarm's whereabouts by printing the point (default scenario, see Chapter 6). This is implemented in the movement method, when the swarm reaches its target location, the user is acknowledged that the swarm passed through this point and is heading to the next one. When the drones arrive at the transmitter, the program tells the user that the packets have been picked up.

When a drone is incapacitated, a printing method is implemented inside the `removeDrone()` method to inform the user which drone has been incapacitated.

```
fullData = information.getWholeInfo();
if(fullData.equals(receivedData) == true)
{
    System.out.println("Mission passed. All packets have been delivered.");
}
else
{
    fullData.removeAll(receivedData);
    System.out.println("Mission failed. Missing packets: " + fullData);
}
```

At the end of the mission the system prints out the received packets (filtering duplicates), and based on whether the program has been able to reconstruct the whole message, the program

prints an appropriate output, informing the user if the mission is passed or failed. When the mission fails, the missing packets are also printed using the snippet of code shown above. This is achieved by removing all elements from `receivedData` that match elements in `fullData`.

7.3 Summary

This chapter discussed how the message is being formatted, depending on circumstances. In order to incapacitate drones the program uses two separate ArrayLists: one which stores the drones, and another one which contains all the data stored by the swarm. The process of drone elimination is unique since the enemy units are defined in another class. The swarm makes the coordinates of each drone accessible for the enemy units by passing these coordinates to the Playground class. From this class, the enemyUnit takes these values and calculates the optimal target, which once found, is being fired upon. The enemy unit has a success rate of hitting a drone by each shot fired, if it succeeds, it passes a Boolean and an integer value – the Boolean is to determine whether the drone is shot, whereas the integer defines the ID of the targeted drone. Since the Swarm class is instantiated in Playground its methods are accessible, therefore these values, the Boolean and integer, are passed from the Playground to Swarm where they are used as parameters in the appropriate methods, such as the one that removes a drone. The swarm class then executes the code inside the method, removing a drone and its corresponding data stored from the ArrayLists.

The project required several tests during its development which were evaluated. Chapter 8, Testing and Evaluation describes the tests I performed on classes I implemented, and tests we performed on the whole program. The chapter will also point out specific concerns regarding the program which required attention in particular.

8 Testing and Evaluation

8.1 Testing During Development

Various tests and refinements took place during the development of the program. Each time a bug or malfunction was present, we halted the process to fix the problem and only continued when the issue was solved. This allowed us to keep the program relatively bug-free, and we knew disregarding those problems would cause more errors and bugs.

8.1.1 Testing Classes I Implemented

I tested all the classes I implemented to ensure they would not break the program. Once it was certain that the code did not include any unwanted code and was functioning as intended, I uploaded them on the cloud storage.

8.1.1.1 *The Swarm Class and Its Instance in the Playground*

The development of the swarm class composed of multiple steps since it has multiple purposes. The first step was to make a single drone appear on the battlefield. This gave a basic idea of how a single drone should appear, the size of the drone and its colour was adjusted to make sure it is visible properly.

The next step was to make this single drone move from one point to another. This required a lot of testing to ensure the speed of the UAS was appropriate. The function which allowed the drone to move was very basic at this point. Different values were assigned for variables that were used in the function to find the optimal speed for the drone.

Then multiple drones were created around the single drone which became the leader. The mathematical formula was described previously, but initially the drones had a fixed distance from the leader, meaning they were all in one line, not in an organised formation. As soon as this was achieved the objective was to create rings of drones around the leader. The mathematical formula was worked out at this point, but the outcome was unexpected, therefore the code had to be improved further. To achieve the desired formation, the formula was expanded and refined using different variables and values. This required a strong understanding of the formula – I had to know what exactly happens when the program calculates the coordinates. I tested the equation by inserting values of the number of drones, number of drones in a ring, number of rings, and remainder. These values were also multiplied by counters of each loop – all done separately. Finally, multiplying 90 degrees in radians by the counter of the number of rings yielded the desired outcome.

In order to meet the requirements, the swarm had to go through multiple destinations on the battlefield. The movement method had to be expanded by allowing it to accept multiple coordinates. However, when the swarm changed its trajectory, it gradually slowed down which was caused by the incorrect handling of values of distances between each coordinate. Negative values weren't considered, but both James and myself started to work on a solution. Eventually, James proposed a solution therefore I adopted his code.

Another responsibility the swarm had was the transmission of packets. The message consisted of 150 packets which had to be distributed evenly between the drones. Fair and even distribution was hard to achieve because the program had to remain realistic, meaning the user still had to be able to pass the mission while maintaining a chance of mission failure. The number of packets to be erased from each drone's memory had to be optimal. To ensure this, different equations and values were tested. Initially, the value was fixed but then it was

improved to change dynamically, depending on the number of available drones. This improved the chances of passing the mission because the number of packets missing at this point was more optimal. To further improve the code, the equation itself was improved. Sometimes the size of the array in which the drones were stored caused an issue – the index was out of bound. To achieve the most optimal way of formatting the message, I decided to consider the number of available drones and the number of packets in the full message. These variables were divided then the result was applied in a loop that eliminated a number of packets equivalent to the yielded outcome. When first running the code, the mission always succeeded, but when I made a few changes I realised too many packets are missing, therefore the code needed some optimisation.

8.1.1.2 Other Classes

The Base, Receiver and Transmitter: these classes required aesthetical optimisations, their sizes had to be carefully defined. As for the base and receiver, my aim was to place them in the corner, in a location where they perfectly fit. Their sizes were the same – their centre locations were 50 units from the border so their circumference were 50 units long which allowed to perfectly fit them. The transmitter was significantly smaller than the base and receiver.

Information and Drone Class: initially, the full message was created in the Swarm class, but the method had to be separated. There were issues with passing an ArrayList as a parameter, a solution for this was declaring ArrayList variables in the Swarm class as local instead of global variables. Finding this solution took a long time, I didn't think that global variables caused the issues – relocating the variables was my last option to find a solution for this. Whenever I tried to debug the program, the Swarm modified the variable in Information which stored the whole message, so it was not accessible after methods were called that formatted the message in the Swarm class. This problem wasn't present in Information but it affected it indirectly. To solve this, a number of tests were performed on the way the Swarm class handles its variables. An initial approach was to create more temporary variables to store data, but this approach failed.

The drone class didn't require too much since only some variables were used to store some values passed on from the swarm.

Dot Class: the class acquires some coordinates from the Swarm class, using a get method in Swarm, and it sets its own coordinates using set methods. This required synchronisation, initially the set method was called earlier than it was supposed to be, which resulted the marker to appear at the previous target location, instead of the next one. After revising the code, the problem was found quickly and the error was fixed shortly by moving the set method after reaching the target coordinate, and not before. This ensured that the marker was always ahead of the swarm.

8.1.2 Testing the Program as a whole

The program had to be tested in general – both its functionality and performance had to meet a level of satisfactory. We had to make sure all classes interact with each other in an intended way.

8.1.2.1 Performance

When mentioning program performance, we consider factors such as resource handling and operation in different environments. A major issue we encountered was the depletion of resources when the enemy was firing at the swarm. The problem occurred in the EnemyUnit class. James informed me about the issue, which was when an enemy tried to shoot a drone, the whole swarm was continuously scanned by the EnemyUnit class – the printing function also slowed the process down drastically. We decided to propose solutions for this problem, and we established that the swarm had to provide the coordinates of each drone so that the enemy can determine the location of each drone and fire at the closest one without depleting the system's resources.

The problem occurred when the shooting function was implemented. Once the solution for the performance was tested and it met the desired expectation, the function itself had to be tested further (more details in 8.1.2.2 - Functionality).

8.1.2.2 Functionality

The emphasis was on how realistic the program was. First, we tested the functionality of the enemy unit. Soon we found out the enemy fires at the swarm at an excessive rate. This was fixed by James by implementing a timer which was thoroughly tested and adjusted. Another factor that was considered when testing this was the rate of success – we had to make sure the drones were incapacitated as intended.

The user may decide not to use all 30 drones when starting the mission, therefore we had to test the program in different scenarios to ensure the mission was still possible to pass, also, the program shouldn't start when the user decides to have more than 30 drones in the swarm. To make sure the program won't run with more, than 30 drones, I capped the size limit of the ArrayList in which drones are stored to 30.

When the swarm starts the mission with less than 30 drones, the chance of success of completing the mission should decrease, based on how many drone are present. The probability of passing the mission is enhanced by the algorithm which distributes the packets between the drones. When less drones are available, less packets are removed from when formatting the message. This way the enemy still wouldn't be able to acquire all packets, but the chance of passing the mission is significantly reduced because there are less drones available from the start, meaning the swarm is at greater risk from being completely incapacitated. Different scenarios were tested and after evaluating the results, we decided to adjust some of the numbers until we were satisfied with the outcome.

8.2 Final Tests

The final tests entailed testing the program as a whole. Our aim at this point was to ensure the input file is working correctly, the simulation is realistic and satisfying, and the modules of the program work together as intended. Since we tested the program throughout the development process, at this point there was nothing significant to test or debug. The only feature that changed was the way the program prints the messages – we decided to eliminate certain printing functions and implement other ones so that the output would be more meaningful.

8.3 Evaluation

I needed to run the program multiple times using different inputs to create different scenarios. The test entailed running the program with different amounts of drones and each scenario was simulated five times using the same input then calculating the mean of the outcome. The numbers of drones used for testing were 30, 25, 20, 15, and 10. The table below shows the complete result of the tests, Figure 8.3a shows a graph with a line of best fit for these tests. The value of x is the number of drones when the mission starts, the value of y represents the number of drones which arrived at the receiver.

<u>x</u>	<u>y</u>					<u>Avg.</u>
30	19	22	20	20	22	20.6
25	18	16	18	17	17	17.2
20	11	11	14	10	12	11.6
15	9	7	6	9	6	7.5
10	0	1	3	1	0	1

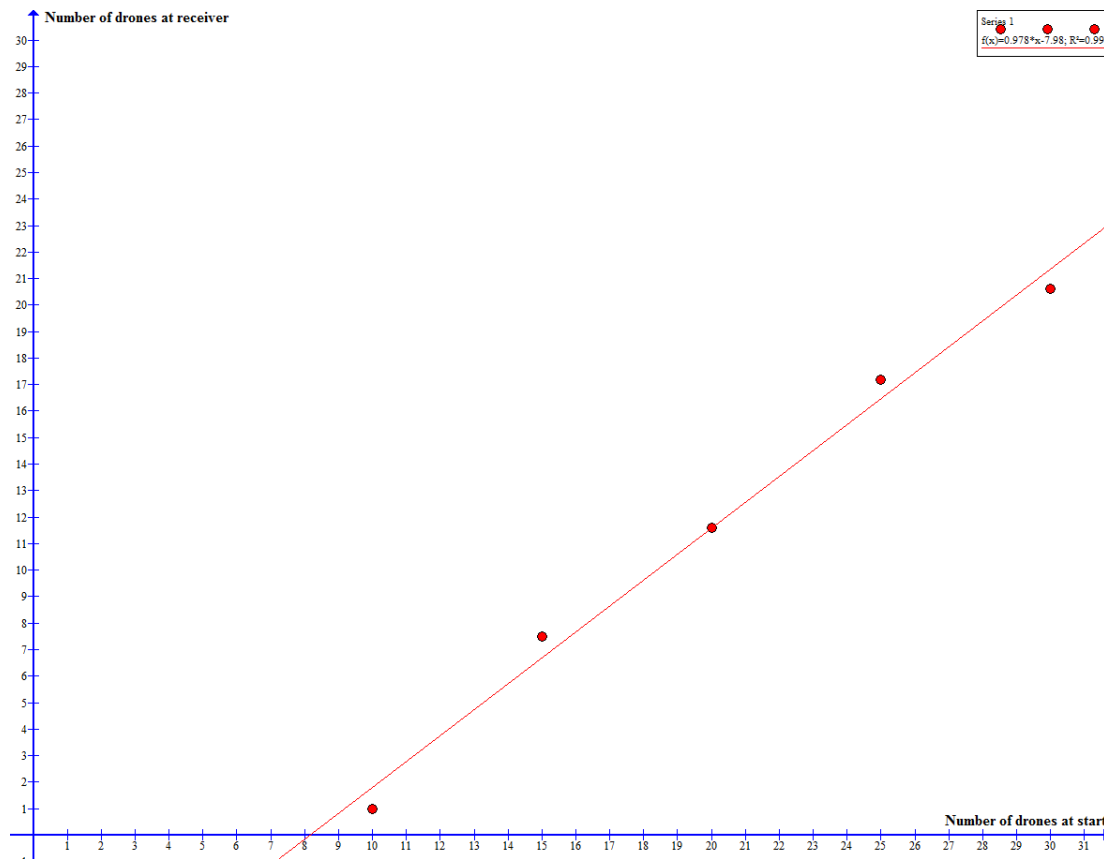


Figure 8.3a: the average of the test results is shown on the graph. A line of best fit is applied which can help predicting the number of drones expected to be arrived at the receiver.

Using this set of data, I analysed the percentage of drones incapacitated in each scenario. A table is also provided to show the results of the tests and Figure 8.3b shows a graph, provided with a line of best fit. The value of x is the number of drones at the start of the mission, the value of y represents the percentage of incapacitated drones. Analysing the losses of drones in percentages can help predicting the mission outcome, since the higher percentage we have the more likely we are to succeed.

<u>x</u>	<u>y (%)</u>					<u>Avg. %</u>
30	63.3	73.3	66.7	66.7	73.3	66.7
25	68.8	64	68.8	68	68	67.4
20	55	55	70	50	60	58
15	60	46.7	40	60	40	49.3
10	0	10	30	10	0	10

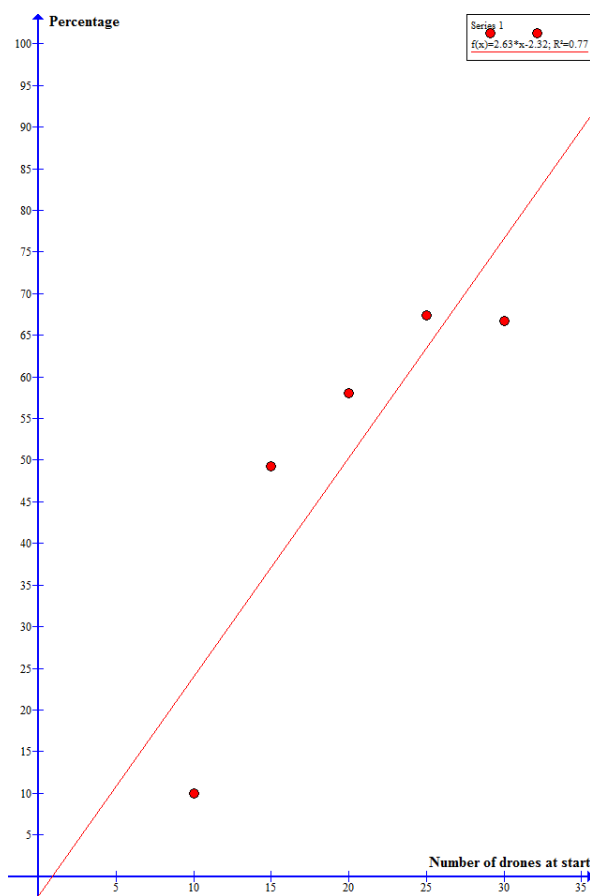


Figure 8.3b: the average of the percentages is shown on the graph. The line of best fit helps to determine the mission outcome.

8.4 Summary

Continuous testing during a development is crucial for achieving the intended goals and requirements. In this chapter, it was revealed how we tested the system, but also why it needed to go through some refinement processes. Initially, the aim was to implement and idea, ensure that it compiles and runs. This had to be taken to another level in order to meet the requirements, for example when swarm was able to move between two points that meant the movement method was somewhat implemented, but the swarm had to be able to follow a path, consisting of multiple vertices.

9 Conclusion

This chapter draws conclusions from the project, taking the project aims into account. The chapter also discusses possible improvements, program performance, code elegance, and extra features that may be added to the program. Finally, the chapter described the most important learning outcomes.

9.1 Project outcome

Generally, the program reached our level of satisfaction. All the “*shall*” requirements have been met, as well as most of the “*should*” requirements. The sections are particularly focused on what could have been improved on the program, including the “*should*” requirements. Additionally, extra features are discussed which could make the program more polished and pleasant to use. Section 9.2 introduces some of the requirements that have not been met and it also describes some possible solutions, for the complete set of requirements, see Chapter 1.1.

9.1.1 Review of Requirements

At the beginning of the project James and I divided the work evenly after we produced an initial design of the product. Studying the design helped us to distribute the work evenly. This section describes the requirements concerning myself and both myself and my partner.

9.1.1.1 Requirements Met – Individual Goals

I had a set of requirements for the Swarm, the Path, and the Communication System. The program is able to display up to 30 UAS units, which was achieved by capping the size of the ArrayList which stores the drones to 30. Each drone is described by speed, power consumption, and energy storage – there are variables which define these features. The swarm takes off from a single base, which is located in the bottom left corner on the map, then it heads to the transmitter where the packets are being picked up and then delivered to the receiver. If a drone is incapacitated, then the swarm takes a new shape which depends on the number of drones available in the swarm.

Although the z-coordinates are not being used, the path is defined in 3D. The path consists of multiple vertices and the trajectory between each of these points is a straight line.

There are three actors in the communication system: the collection of drones, the transmitter, and the receiver. The transmitter and receiver have a fixed position on the ground. When the swarm arrives at the transmitter, the transmitter creates the whole message, formats it, and then assigns it to each drone in the swarm. The drones then store the packets and attempt to deliver it to the receiver. The full message is composed of a fixed number of packets, which is created and formatted when the swarm arrives at the transmitter. When the drones approach the receiver, the receiver tries to rebuild the whole information using the data stored by each drone.

9.1.1.2 Requirements Met – The Program as a Whole

The simulator was developed using Java. At the beginning of the project, we discussed possible options for open-source software development and we opted to use Java. All the elements of the program are represented in 2D which provides convenient visuals since the user is able to see everything from above. The size of the playground is 1000 by 1000 pixels,

each pixel represents one meter. The user is also able to input desired parameters using a single text file.

9.2 Possible improvements

There is a range of possible improvements could be made to enhance the usability of the program and improve user experience, these are the following:

- Packet redistribution within the swarm when a drone is incapacitated
- Smoothly display changes in swarm formation
- A text file provided by the program which describes the mission outcome
- Enable the user to toggle between automatic flight and manual flight
- Enable displaying a 3D environment

The project aims simulate an innovative method, which is to use UAS devices for relaying information. Our scenario involved a number of drones, a number of enemies, two bases, and a transmitter – simply speaking a battlefield. Since the mission takes place on a battlefield, the drones may become incapacitated, but when they already have stored information, the packets are not redistributed to enhance the chance of mission success. A possible way of adding this feature is to implement another method, similar to the one that formats the message at the transmitter, and call this method inside another method, the one that eliminates a drone. This method would use the identical algorithm used for formatting the message since the way of formatting the message involves the number of available drones.

An aesthetic improvement would be to create a smooth animation when a drone becomes incapacitated and the swarm takes a new shape. Instead of the drones changing positions immediately, the whole swarm could slow down while the drones relocate. The user would be able to see the drones move to their new position in the swarm. This requires some modifications to the movement method inside the Swarm class as well as implementing an additional method. The extra method would modify the movement speed of the swarm temporarily, and once the drones are done relocating it would reset the speed to its original value. An additional purpose of this class would be to keep the speed of the currently relocating drones and continuously update their coordinates. All drones have their positions defined based on the leader drone, which is located in the centre of the swarm. The leader follows the path while the other drones in the swarm follow the leader. To enable smooth animation for repositioning of the drones, new target coordinates have to be set for the drones that are to relocate themselves. This method would be similar to the movement method, however it would work on relocating drones only.

Another possible improvement would be to use a text file which outputs the mission details. The program currently uses the console to print messages, which is not very user-friendly. The program should be able to write the mission outcome to a file which would provide a summary. The text could contain details, like how many were shot down during the mission, locations where drones became incapacitated, packets stored by each drone when arriving at the receiver, reconstructed message (a “raw” result of the mission), and a final sentence which states whether the mission was successful or not by adding “Mission passed.” or “Mission failed.” – in case the mission fails, an additional line would inform the user about the missing packets.

A possible requirement stated that “the simulator *should* allow a togglable manual flight mode.” – this would make the simulator more engaging. This would require the

implementation of action listeners and modifying the whole structure of the code inside the Swarm class. Action listeners in Java allow the program to take inputs from hardware input devices, such as a keyboard or a mouse.

It requires threading and complex mathematics to create an environment in 3 dimensions. The program uses a Java tool called Swing, other options are discovered in section 9.1.2. However, it is not possible to create 3D graphics when using Swing only. Another tool called JavaFX has to be integrated into Swing to allow creating the 3D environment. Assigning a hotkey would allow the user to toggle between 2D and 3D – this could also be done implementing action listeners.

9.2.1 Program Performance

The program performs as intended, since it does not have any latency issues nor bugs that prevent the program from functioning properly. In Java, the printing method has to be used sparingly because it uses a hefty portion of the system's resources. Because of this, we had to avoid having too many printing methods, and these were only used when necessary. We encountered a problem with this: the enemy unit called the print method excessively which drastically slowed down the program.

We decided to use Swing to visually represent the playground. The reason for not enabling 3D in our program is because it would have taken a substantial amount of time to rewrite the code. Integrating JavaFX would have required us to strictly focus on the Playground class due to the need for handling threads and complex mathematics. The structure of the class would have significantly changed as well since new variables and methods would have required implementation. Hence we decided not to change the Playground class, which was already performing well with Swing.

9.2.2 Code elegance

During the development of the program, our aim was not only to produce a program that meets the project objectives and requirements, but also to make sure the code is readable, elegant, and efficient. Whenever I worked out an algorithm, I started working on the solution using pen and paper, then I translated it to code. I also made sure the comments clearly describe the purpose of each method and algorithm.

To promote readability, I also named the variables in a reasonable way so that they reflect their meaning, for example a variable `xPosition` stands for the current x-coordinate of the leader drone.

Another factor that matters in code elegance is the tidiness of the code. I intended to write the program using as few variables as possible as well as laying out the lines of code smoothly – for example using line breaks where necessary, or reducing the number of if statements by merging all necessary conditions in one line.

9.2.3 Extra features

The functionality of the program could be expanded by introducing borders on the battlefield. These borders would prevent the drones from leaving the scene and become invisible for the user – for example the size of the playground is 1000 pixels by 1000 pixels, but the drones are not restricted to stay within this range. Checking the input coordinates may help preventing this, if there is a coordinate that's beyond the boundary, cap its value to the lowest or highest acceptable number, for example instead of having the coordinates (20, 750, 35) the

program should cap the x-coordinate to 50, for instance, so the target location would become (50, 750, 35).

Another, more complex yet rewarding feature would be to allow the user to define the target locations by clicking on specific areas on the map. Before running the mission, the program could open the map of the battlefield, inviting the user to define their locations by clicking. This would also make use of action listeners, as well as ArrayLists, which would store the identified locations' coordinates. Once the user finished identifying the target locations, they could press a key, space for example, to start the mission.

Currently, the program only displays one window. To extend the GUI, another window could be added. The purpose of this would be to allow the user to enter coordinates into specific fields from which the program would take the entered values as input parameters. Other than the coordinates, the user could be able to define speed, energy storage, number of drones, and number of enemies.

9.3 Learning outcomes

There is a wide range of skills I have improved, both in technical and non-technical areas. The project provided us freedom to approach and solve a problem in order to meet the requirements.

9.3.1 Improvement of Technical Skills

There is a range of skills I have improved during this course. More specifically, regarding programming in Java, as follows:

- Handling ArrayLists
- Using Swing
- Implementing mathematical equations

The course helped me to gain a deeper understanding of how ArrayLists work in Java, since one of the major problems I encountered was regarding this specific type of data. I became more confident in manipulating ArrayLists using the provided methods, such as `arrayList.remove()`, `arrayList.add()`, `arrayList.size()`, and many other ArrayList-related commands.

I also gained a greater knowledge of Swing, a tool which allows to display objects on a graphical user interface. A class which we co-developed called Playground required me to implement my own objects. While developing this class I gained a better understanding how to display or remove objects.

The program involved some relatively complex equations. One of them is responsible for maintaining the appropriate shape of the swarm, the other one is used to format the message in the communication system. I found it easy to work out these formulas on paper, however, the implementation was more challenging. None of these equations resulted in the desired outcome for the first time, but as I was improving the code I became more and more confident, until I achieved the desired outcome

9.3.2 Improvement of Non-Technical Skills

This section is concerned about the experience gained in non-technical skills, such as:

- Decision making
- Teamwork
- Communication
- Planning and design
- Problem solving

The requirements of the project were provided, but it was up to us how we produce a program which would meet these expectations. This gave us freedom to make our own choices. One of my skills which drastically improved during the course is decision making – a critical skill in computer science. I was aware of the range of options available to solve specific problems. During the earlier stages, I had initial designs and implementations for some of the problems which were revisited later on and modified to make these solutions more suitable to the future content. As I progressed with the coding, whenever there were multiple solutions available, I considered every single one of them and outlined the possible outcomes. This way I was able to find an optimal solution.

Teamwork is another key skill which I improved. I have not had too much experience in working in teams before. This skill consists of many other key skills, such as communication, which is crucial for an effective product delivery. Throughout the course I maintained a good understanding with my colleague and we kept each other updated as much as possible, for example when a new piece of code was available we acknowledge each other. Whenever we encountered problems we reported them and sometimes we solved these co-jointly.

Planning and design is also an important skill to have. I also improved in this area, since the project provided such freedom, all the planning and design were in our hands. I had to carefully design the classes I needed to implement, which is important since the program is built on these basic designs.

Problem solving is a necessity, a skill which I have been constantly developing and improving over time. Since the project provided freedom in terms of achieving the requirements, the majority of the problems I have encountered was created by myself. Let's take the communication system as an example: working out the algorithm was not the tough part, instead it was the implementation – some of the functions did not work as intended, so I had to find the errors and problems which caused the malfunctioning of the program.

References

- 1: <https://www.aeryon.com/press-releases/libyanrebels>
- 2: Yoo, Christopher S., "Moore's Law, Metcalfe's Law, and the Theory of Optimal Interoperability" (2015). Faculty Scholarship. Paper 1651.
- 3: <https://www.parliament.uk/business/committees/committees-a-z/joint-select/human-rights-committee/inquiries/parliament-2015/uk-drone-policy-15-16/>
- 4: <http://www.birmingham.ac.uk/research/perspective/drones-and-targeted-killing.aspx>
- 5: <http://www.bbc.co.uk/news/world-south-asia-10713898>
- 6: <http://www.upi.com/Archives/1986/05/29/Rights-group-reports-on-abuses-in-El-Salvador/6787517723200/>
- 7: <http://tweakyourbiz.com/technology/2015/09/18/open-source-programming-languages-basic-overview/>
- 8: <https://www.python.org/about/>
- 9: <http://www.bell-labs.com/usr/dmr/www/chist.html>
- 10: <http://www.thecrazyprogrammer.com/2013/07/what-are-advantages-and-disadvantages.html>
- 11: <http://www.infoworld.com/article/2887974/application-development/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html>
- 12: <https://www.ruby-lang.org/en/about/>
- 13: http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.performance/advantages_java.htm
- 14: http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html
- 15: <https://www.quora.com/What-are-disadvantages-of-Java>
- 16: <http://www.businessnewsdaily.com/9276-commercial-drones-business-uses.html>
- 17: <http://foxtrotalpha.jalopnik.com/exclusive-photos-avenger-the-camry-of-advanced-combat-1571300984>
- 18: http://www.militaryfactory.com/aircraft/detail.asp?aircraft_id=757
- 19: <http://www.proxdynamics.com/products/pd-100-black-hornet-prs>
- 20: <https://www.techwalla.com/articles/the-disadvantages-of-ruby-programming>
- 21: <https://www.quora.com/Which-is-better-Java-or-Python-for-making-an-application-GUI-with-cool-look-and-feel>
- 22: <https://www.quora.com/How-can-I-create-GUI-using-C-programming>
- 23: <https://www.quora.com/What-are-the-pros-and-cons-and-uses-of-the-major-programming-languages>