UNIVERSITY OF BIRMINGHAM

Collective Behaviour in StarCraft 2

Project Report

School of Computer Science
MSc Advanced Computer Science

Sotiris Loizou - sxl1028 - 1966658
Supervisor: Dr Eliseo Ferrante
Project Category/Topic: AI

10th of September 2019

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that we have neither committed plagiarism in the completion of this work nor have we colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Academic Integrity (Appendix A)

Abstract

Swarm intelligence is a broad term that originates from nature. It is a behaviour that many animals exhibit when grouped together. The animals that tend to flock together are relatively small in size and very vulnerable on their own. Some type of animals that tend to collectively group are insects, small birds and fish. The rules which these animals follow can be described as simple and efficient. An example of such a simple rule in an ant colony is allowing a distance between your neighbouring ant while moving at any direction. The efficiency can be observed while the ant makes small adjustments to obey the rule which translates to small physical movement. This can be noticed on an individual level for every member in the flock. The fascinating aspect is the collective behaviour that emerges when the whole colony is observed in its entirety. This type of collective behaviour is documented in a variety of social species which all obey different rules.

This paper mainly received inspiration from prior work that occurred which focused on collective behaviour exhibited by a flock of robots. Their collective behaviour is going to be transferred and implemented on real-time strategy game called StarCraft2. The specific game has been picked due to the unique environment as well as the lengthy refinement of the game. This is a first fundamental step towards introducing collective behaviour in StarCraft. There is no prior recorded attempt to implement or even apply any of the collective behaviours in StarCraft. This is the main reason of introducing swarm intelligence to try and explore a new possible avenue. The first version of the game has been launched in 1998 and at this instance the second version still receives updates. Both versions of the games have an e-sport scene where professional players compete for titles. So far deep mind has produced exemplary results by creating e-sport level agents that can play the game. Agents that managed to win against real professional StarCraft2 players.

Acknowledgements

I would like to start by mainly thanking my supervisor Dr Eliseo Ferrante. He was available through the whole process. I personally appreciate that I had the opportunity to investigate and explore the field of collective behaviour which was previously unknown to me. Most importantly his prompt replies when communicating was a vital part of overcoming obstacles and issues that arose. I would also like to express my gratitude towards my course mates that helped by having meaningful discussions about our projects. The IT department along with the reception in the computer science school was of great help when setting up the skype calls with the supervisor. In addition, the time spend together also created a good friendship. I also deeply appreciate the help I have received from the StarCraft 2 AI in discord Community. Finally, I would like to express my very great appreciation to family that encouraged me in times of despair and confusion.

Table of Contents

1. Introduction

The platform that has been selected to embark on the upcoming endeavour is StarCraft2. One could simply ask what would have been the reason of such a decision. There are various simulations environments that have been created for this exact reason. One of the most known one is ARGoS in the field of robotics. This is a multi-variate simulation that simulate a variety of robots as well as the real-world physics to some extent. ARGoS has been proposed and created recently relatively to the StarCraft franchise. The release date of StarCraft is around the first annual quarter of 1998 which makes it a 21-year-old game with one expansion. The second version later called StarCraft 2 with 3 consecutive expansions has been released during the second quarter of 2010, making StarCraft2 a 9-year-old game.

The game itself is a real time strategy game that is known for its substantial skill-oriented theme. This distinction is quite important since it allows the game to mature both its agents as well as the environment. StarCraft2 has 3 factions with a plethora of unique units which had to be balanced to avoid any unfair advantages that one faction can have over the other while maintaining different viable playstyles. This is evident due to the e-sport scene that has been created for both games. Professional players pushed the game to its limits trying to find any available exploits as well as refining their strategies. This provided blizzard the creator to tweak the game to fix any bugs and unit aspects to promote an overall balanced game. Introducing collective behaviour may also reveal bugs and assist for further development of the game. This can be assumed by the scenarios that can be produced by a collective behaviour that was not possible previously.

Based solely on the aspect of the years that this game has been refined, it seems like a promising candidate to attempt to implement some key aspects of swarm intelligence. An interesting note is the fact that there are no prior pursuits to develop any type of collective behaviour mechanism in StarCraft2. The idea of genuine work on this platform is another motivating influence. More specifically, this project attempts to implement collective behaviour within a group of units which acts as a flock/swarm. It is important to reinforce the individuality of every unit, similarly to the nature-based behaviour. A mechanism must be created to control all the units as well as the creation of the environment. The most important aspect is to actually generate the behaviour and be able to finely tune the behaviour. In nature this already observable due to the years of natural selection.

The complexity of the game is noticeable both in units and environment. The later was a contributor factor to considering the game. As mentioned previously there are three factions zerg, protoss and terran. All functions have different units that are only available in that faction. What is more all factions have different style buildings which are necessary for the games playing progression. The environment has various distinct attributes which mainly focus on terrain details. To begin with there is ground elevation which allows the ground to be on different height levels. Additionally, there are destructible environment objects which there only utility is to block available paths. Another unique characteristic that the game provides is called the fog of war. This is an area that is not visible to the units if no units of the same player are physically present in that area within a specific radius. Additionally, this characteristic can be observed in tall foliage where the visibility radius is based on the foliage spread. All the above characteristics favour the aspect of creating the required behaviour.

2. Literature Review

The aspect of swarm intelligence that this project mainly focuses on is collective behaviour in self-organising groups. This is a necessary distinction to be made since swarm intelligence is a broad term that has a more abstract and larger scope. The main inspiration of collective behaviour has been the nature. Reynolds has expressed this behaviour in different terms compared to the old existing scripted paths that was the norm. The new adaptation aids a simulation towards a more realistic representation of the natural flocks. One of the first steps was the introduction of every agent in a flock as a unique individual. This individual has a relative perception of its surroundings within limits. Every individual act based on its own understanding without a direct access of information which are out of its scope. Reynold focused on a specific example regarding a flock of birds, but this can be used on any self-organising groups.

Such groups found in nature are fish, birds and insects which can be described as physically tiny. This is the one of the reasons that resort to grouping together for better chances of survival. Reynolds expresses the problem that arise from the scripted paths of all individuals. The distinct issue is the fact that no two different particles/agents can have the same scripted path all must be unique. This subsequently creates the problem of storing all these paths as well as generating of all of them. The animation in the simulation of scripted agents produce rigid predictable motions. This type of motions does not seem natural but rather a step towards automatic robotic flow.

Instead the paper describes a different approach that focuses on the individual level of every agent and its interactions with all its neighbours. This promotes a more natural representation while the system becomes more efficient since its rather simplified and object-oriented. The solution proposed is derived from a particle system that focuses to model fire, smoke and clouds. The system simply has a set of a certain life cycle. Every particle is created then aged until it dies off and fades out. Every particle has a state that can be altered by changing its characteristics (colour, velocity, location). There are other problems that arise by using bigger three-dimensional object that have different shapes. This mainly relies on the environment in this case the simulation itself to be robust. The simulation should deal with object collision and the path finding of the agents within the environment. In StarCraft 2 this problem is negligible since the game environment has sub-systems that avoid and combat these issues.

The actions of every agent have both a direct and indirect effect on the group in its entirety. The direct effect is being produced by the neighbouring agents of a focal agent. A focal agent is the agent that is active and about to do an action. Thus, the focal agent is affected by its surroundings when is about to update its state. The state of all its neighbours influence the new state of the focal agent. A hypothetical example can be a drastic velocity change of an agent due to a thread. The information about the thread will not pass to the resto of the group, the drastic velocity change will be passed and effect the group's overall movement speed. The indirect effect can be described as rippling effect that is past on wards from neighbour to neighbour until it reaches the focal agent in that instance. The indirect effect can be hardly described or predicted, especially in environments with a variety of alternating aspects. For instance, some aspects of the environment are physical objects that are not included in the group but physically affect the movement of the group. Furthermore, the size of the group can also introduce subtle variability to the collective behaviour.

The simulated flock system in Reynold's paper describes attraction, alignment and repulsion. The attraction aids the unity between first degree neighbours which subsequently is the mechanism that keeps the group together. This is achievable due to the rippling effect described previously that every agent's action has to the whole group. The repulsion mechanism is a mechanism to prevent agents from coming close and collide. The repulsion is important to avoid any situations where agents end up pushing their neighbours out of the way due to the minimal distance between them. This type of movement can be catastrophic in a group where there is not enough space available for all agents to move accordingly. Alignment can be described as a secondary feature nowadays since most simulations predict alignment based on the agent's movement path. During Reynold's years it was another aspect to be considered and had to be updated for every agent. In a physical world this must be considered as well. An example is in an autonomous flock of robots where the alignment must be dealt with appropriately.

Next is the introduction of a new characteristic known as informed agent as promoted by Couzin et al (2005), with the concept of informed and uninformed individuals. The main purpose of the informed individuals is to have an intuition or a preference in movement. This preference can be for instance the path to a goal or the next step towards a goal. Every individual has a vector which indicates the magnitude and direction of an individual. The informed individual has two vectors one for the neighbourhood and one for its intuition. The uninformed individual has only one vector regarding its neighbours. These vectors are

calculated at every time step by all the members of the flock and then each individual alters its position based on the vector(s).

What is more a balance must be defined between the preferred direction and its social interaction with a weighting system. The social interaction emphasises on the neighbours' dynamic with the focal individual. The weight will have to proportionally select a percentage for both the social and preferred vectors. This subsequently creates a new vector by combining the two vectors based on the weight specified for each one. In addition, the number of informed individuals must be balanced in order reach a goal with a minimum number of informed individuals. In the paper Couzin argues that finding a balance in how many informed individuals should exist if of higher importance than balancing the weight for the vectors.

Upon further investigation regarding StarCraft2, there is no prior attempts to apply collective behaviour at the game. DeepMind's attempt to create an agent that can play the game is one of the most noticeable and applaud pursuit to apply artificial intelligence. This focuses only actually playing the real time strategy aspect of the game. The results are quite surprising since the agent achieved to master one of the combinations of factions' duels. The agent can play protoss versus protoss to a high level. An event was hosted by DeepMind where two professional players compete against the agent in a best of five games. The agent managed to win against both. Their approach is to use mainly deep reinforcement learning to train the agent. Additionally, they used co-evolution in order to aid the improvement process that the agent had to undergo. Alongside co-evolution imitation was used to learn from recorded games while supervised.

3. Project Specification and Analysis

3.1 Description

This project has as a main focus to introduce swarm intelligence to a new platform. As mentioned previously the platform selected will not be a conventional simulator but rather a strategy game. The main inspiration is derived by flocking robots that has been introduced by Ferrante et al (2016). The paper examines how the repulsion and attraction model introduced by Reynolds can be applied to a group of robots. Throughout the paper a special consideration is made due to the fact that real robots and simulated robots do not exhibit the same results. This is evident due to the fact that real physics (environment) is not a complete match to the simulation. An experimental approach is used to see how the two environments differ. A rigorous method of creating specific experiments and running to compare results. The main aspects of the vector for each robot is attraction, repulsion and alignment. The game does not require any additional alignment since the environment manages and corrects alignment. This is not the case as expressed in groups of robots, especially in real world scenarios.

Proximal control is the idea of allowing the robot/agent to maintain a specific distance from its surroundings. The next aspect discussed is the motion control which can be magnitude dependent or independent. This means that the magnitude of the vector of each robot has to be considered when calculating the displacement. The game at this point introduces the first limitation which is the static speed that units have and cannot be altered easily. The speed can be changed in form of upgrades, but this only occurs when the game is played as intended by building specific facilities and following certain upgrade routes. Another limitation is the fact that once upgrades occur, they cannot be reverted which results in specific type of units having a permanent higher movement speed.

3.2 Objectives

There is a list of abstract at this point objectives that must be fulfilled afterwards by the solution in order to accomplish the implementation of the collective behaviour in the game

- Creation of specific environment with specific scenario
- Ability to control every individual agent
- Isolate every agent by encapsulating information to every unique agent
- Agents should only have access to their own information only
- Ability to be able to differentiate between agents
- System that has simple naïve group behaviour
- Another system that has collective behaviour
- Creation of the attraction and repulsion dynamic of the group
- Ability to control different aspects of the behaviour
- Scalability prospects for different size of groups
- Ability to measure certain metrics to be able to compare

3.3 Objectives Analysis

- Scenarios must be scripted or created that are going to execute within the game's environment. These scenarios must be specific enough that all aspects can be easily monitored and recorded. The scenarios themselves refer to the map characteristics along with all the available units and building arrangements. Furthermore, behaviour that is based on that specific scenario with certain parameters should be reproducible. Control over the environment should be possible to some extend but without any external impositions.
- It is really important to have full control of the available actions that can be executed by an agent, no matter their type. A universal style of control is needed to control specifically the movement of the agent which is the primary action available for the collective behaviour.
- Every agent should be represented uniquely along with all the information needed. The information should only be available when dealing with set agent. All agents should be uniquely identified as well as controlled.
- A simple system that will introduce basic movement with no actual objective which can be used as a reference point also known as a starting point. This probably can be achieved with a system that has randomness to the agent movements.

- An actual mechanism that will implement the collective behaviour as stated by E. Ferrante et all (2016). Further information explaining the parts of the mechanism will be introduced during solution design and then implemented.
- The attraction and repulsion dynamics of the group should mainly emphasise to the vectors that are going to be associated with every agent. The collective behaviour mechanism has both characteristics integrated so as to exhibit correct flocking behaviour.
- Furthermore, the mechanism should have a set of parameters that can be easily altered. These are the main sources of applying and executing experiments along with the scenarios created.

3.4 Available Technologies

There is a plethora of available technologies available to use for accomplishing our objectives as stated above. The environment that is going to be used is now formally stated as StarCraft2. This subsequently introduces the main point of entry for the scenario building objective. The only available way to create a scenario is by using the integrated system of creating maps available in the game, the scenario creator is called StarCraft2 Editor (Blizzard,2010). A pseudo programming language is available to write code which is necessary for the environment. This programming language has both text and graphical representations, the graphical representation seems more effective since options and suggestions are given in a list form while in the text-based system there is no autocomplete and a lack of suggestions or code structure re-enforcement.

Blizzard actually have a mandatory API which acts as an interface between the game and a programming language. The API is called sc2client-proto (Blizzard,2017) which provides direct and specific control of StarCraft 2. The library is written in C++ and has a lot of only primary available controls which do not take into consideration any artificial intelligence aspect. Instead it has been initially built to provide a platform for scripted bots and third-party software that focuses in in depth match analysis and breakdown. Furthermore, DeepMind has built on top of the official API(DeepMind,2017) with an alternative language (Python). This API (pysc2) focuses mainly in aspects of reinforcement learning in order to build an intelligent agent that can play the actual game. This is far from what the project objectives are since collective behaviour has a different setting as well as notion of unit control.

The collective behaviour consists of the environment along with the flock of units while the DeepMind’s system focuses on playing the game with two different agents that play the

game. Reinforcement intelligent agents focuses on controlling all available units while in flocking every unit is unique and self-sustained without having any information outside its neighbourhood. PySC2 has also auxiliary graphical features which represent the game in a simpler form. One can call it a filtering system of what the game screen outputs to a more goal orientated image. A well-known candidate is python-sc2 API which has been produced initially by an individual (Dentosal,2018) but then the software development community aided the progress of the API. Python-sc2 is mainly created to easily write AI bots for StarCraft 2 in Python. The main characteristics as stated by the creator are the simplicity and ease of use of the library. Alternative solutions in different languages have been build such as the ocrfat-sc2cleint (ocraft,2018) which is based on Java. Furthermore, additional APIs have been developed in other languages such as C# (NikEyX,2018) and JavaScript (Node.js) (node-sc2,2018). An important detail that must be distinguished is that all the above APIs have to communicate with the official API provided by Blizzard. Therefore, all the available actions are strictly controlled and set by blizzard in a closed environment.

4. Solution Design

4.1 Introduction

In the previous chapters the problem has been introduced and refined to specific objectives which institutes the project. All the objectives defined during the specification and analysis stage are going to be revisited so as to start designing a solution. For every decision a justification is going to be accompanied. Assuming that this is an initial attempt of building flocking in StarCraft 2 an overall perspective is to maintain simplicity throughout all the design decisions. This is of great importance give the fact that the project does not focuses on playing the actual StarCraft game or building agent that are good in playing the game. Instead, this project aims to use the game as a new platform to apply flocking behaviour of groups. At this point of time limitations cannot be inferred but during the implementation phase an additional section states and describes the limits of the project from a software viewpoint. Upon further analysis it has been established that the solution cannot be written only in one programming language. This is evident from the aspect of creating a scenario and the flocking mechanism that are separate entities. As mentioned previously StarCraft 2 has a strong policy to what can be implemented. A set of candidate classes are going to be stated with all probable needed functionality.

4.2 Flocking Design

4.2.1 Neighbours

- This mechanism is fundamentally based on neighbouring agents. Thus, the set of agents must be found within a range that is parameterised. An additional criterion to finding neighbours is looking for the N closest neighbours where N indicates an integer number of closest neighbours. Functionality is needed for combining vectors so as to update the focal agent's vector based on its neighbours.
- An important addition that has been introduced in this project is the ability to retrieve agents that are stranded away from the group. This can be done by having a limit where the focal agent has to obey at each time step. This limit regards of the number of closest neighbours that are present within a specific range. In case, the focal agent has less neighbours than the limit, it must move back towards the group. It must continue moving back towards the group until it reaches the specified limit. Moving back to the group can be obtained by moving towards the closest agent

based on its current position. This can have a chain reaction where units move towards their closest neighbour which at the end aim to regroup a population of agents.

4.2.2 Loop

- A certain time period has to be established by the game when the agents are updated one by one. This occurs in an iteration where at any point only one agent can be updated. The changes that occur in an agent are then directly projected in the game. An important point is to verify that there the game data along with the software system data are synchronised. The game as mentioned previously is real time strategy so the game's clock cannot be paused or changed. This is due to the fact that Blizzard has a closed game environment as mentioned previously. Therefore, a synchronisation step has to occur at the very begging of the iteration to retrieve all the game's information and alter the software system's information in case of variation.

4.2.3 Movement

- A naïve mechanism has to be built that shows control over the agents in the group. This is the initial step that emphasises mainly on substantiating all the fundamental software-architectural parts that are vital. An abstract representation of the agent must be created that encapsulates all relevant information for an agent. Furthermore, utility style structure that does the main abstract mathematical functionality is separately needed.

4.2.4 Informed Agent

- Functionality regarding the informed agent is crucial since it determines any type of movement. The absence of this characteristic produces a group that is static without any movement. A subsequent characteristic is needed is the social and individual dynamic that has to be considered when calculating the informed agent's vector of movement. The individual aspect favours the next step towards the goal position while the social considers the neighbours positioning. Both are needed in a balance so at to correctly implement the attraction and repulsion mechanism while the group converges towards the goal

4.3 Environment and Scenarios

- Extremely important is also the environment which can more precisely described as a scenario. As mentioned, the game a strategy game which has no direct affiliation with collective behaviour. The scenario is needed in order to create the required

environment so as to apply the collective behaviour. Furthermore, the scenarios are essential for the experiments that are going to be conducted once the system has been produced. All the scenarios need to have a goal that is observable to some extent. This means that the informed agents in the group can find the next step towards the goal. The group itself has to be instantiated with a specific number and type of agents. Randomness has to be applied when selecting the agents' locations within an area as well as the location of the goal.

4.4 Algorithm

Following is a high-level breakdown of the algorithm that has been build in the main class of the project. Further detailed explanation along with the available code is going to be provided during the implementation phase. A set of points are going to follow that describe the overall structure of the algorithm. Every point is going to be developed and described briefly.

- Initialisation phase
- While loop through agents
    - Agent synchronisation
    - Find neighbours
    - Assessing neighbourhood status (TRUE/FALSE)
        - (TRUE)
        - Updating vector
        - (FALSE)
        - Regrouping the flock

4.5 Initialisation phase

This stage starts by selecting a map that has been built to facilitate the collective behaviour. Further details are going to be provided in the implementation phase. Setting up all the parameters that the algorithm needs. Furthermore, the system receives all the information about the agents from the game and creates a copy. A selected amount of the agent(s) are going to be set to informed through random selection. The informed agent(s) receive information about the goal/objective that must be reached.

4.6 Main Loop

The system iterates through all the agent one by one. For every agent its neighbours are found based on a range limit. If the agent is stranded, then its vectors is updated so as to move towards the group. More precisely, the agent tries to move towards the closest neighbours based on its current location. Otherwise, it tries to update its vector according to

its neighbours. In case the agent is informed there are 2 parts different weighted vectors that need to be calculated and combined. These vectors called the social and the individual vectors of the informed agent. The individual vector is a weighted vector towards the next step for the goal while the social vector is the addition of all the neighbours' vectors. In case the agent is not informed then it calculates only its social vector. Next the agent updates its personal vector, then the game changes the agent's location based on the updated vector. This continues until the game/scenario ends, a simple ending for the scenario is reaching the goal. This triggers the game to finish and subsequently cause the collective behaviour algorithm to abort.

4.7 Data and Classes

The information for the project is going to be specified in the following part of the project. An overall picture is going to be presented with all the required data that have been briefly mentioned in the flocking design. This part splits every type of information as the specific data along with its purpose. A temporary class is going to be accompanied with the information with some fields.

4.7.1 Agent Class

Fields

- Tag: A unique identifier that distinguishes every agent
- Type: the type of unit is and whether its informed or not
- Movement Vector: Describes the magnitude and angle for the agent
- Position: a coordinate system with x and y values for the agent
- Target: this is data regarding the next step towards the goal which are only available to the informed marine

4.7.2 Vector Auxiliary Class

Fields

- Magnitude: indicates the size of the displacement
- Angle: indicates a value in degrees from an axis

4.7.3 Utilities Class

Functions

- Find the closest (): This function has as input a set of all the agents along with a number that specifies the amount of closest needed. An agent's index is needed to find the closest based on a specific agent.
- Find within range (): This function is similar to the previous function, but the only difference is that it finds agents that are within a specific radius from a specific agent.
- Add vector(s) (): This function can add two or more vectors together and return back the resulting vector.
- Movement (): this function translates the vector to x and y displacement that the agent has to do based on its current position.
- Euclidean distance () This function calculates as stated the Euclidean distance based on the two agents' points (x and y values)

4.7.4 Main Class

Fields

- Array of agents: this holds all the object instance of the agent class
- Utility: this allows to use all the functions from the utility class
- Set of Parameters: this are numerical values which are bound to the algorithm

Functions

- Naïve movement (): This makes the agent to randomly move around the environment
- Collective behaviour (): This function produces the collective behaviour as described in that algorithm part.

5 Implementation

This part of the report concentrates in the technical aspects of the project. This chapter discusses and states what software solution has been achieved. For any key decisions, arguments are presented to evaluate the possible software decisions that have been made. Furthermore, it aims to provide a detailed description from a software development point of view. It also points out all the limitations and obstacles that have been encountered through the entire process of developing the solution that has been previously stated. The algorithm validation is also an important aspect that proves that the software implemented fulfil its purpose. It ends by introducing the management process of the project which focuses on the weekly workload and its distribution.

5.1 Software Architecture

5.1.1 Fundamental Decisions

At the project analysis chapter there is a detailed listing of all the available APIs (Application program interface). Programming languages range from Java, JavaScript, C++ and Python. As mentioned, the official language used was c ++ to create a direct interface with the game by blizzard. This solution can be described as barebones since it only provides direct control without providing a mechanism that handles the MVC architecture of the system. MVC stands for model-view-controller. The first two aspects are the model and view which are directly implemented by the game. The issue arises when trying to connect the controller part which is the algorithm that was built. An additional non-functional aspect has to be implemented to address this task. Given that the project is not a strongly oriented software development but rather an AI project.

These lead to investigating the workings of other APIs that are extra abstraction on top of the official. The APIs provided by Java and JavaScript where in a very early stage without any major updates or releases during the past few months. This was a warning sign to avoid such APIs since there is little to no documentation on how to use them. They also provide only a limited amount of capabilities, in some cases like in JavaScript it can be described as a code translation to a different language without any new substantial features. On a different spectrum there is the DeepMind's APIs which has a lot of new features but are tight to a specific artificial intelligence aspect. It also adds a few more layers of abstract which are useful and easy to use if the project aim towards reinforcement learning. This is

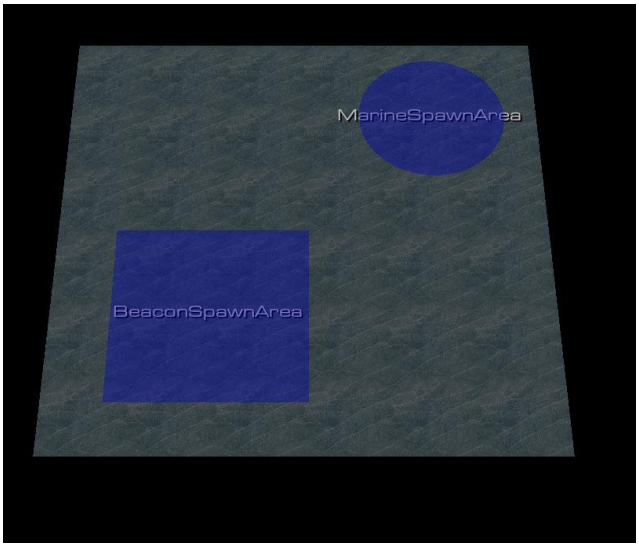
accomplished by providing extra layers of abstraction that can only be accessed via the API. This is quite unfortunate since it does not benefit a collective behaviour project. The most popular language that was used is Python which has been noted in the majority of different projects. Python has a very intuitive almost a pseudo like code syntax. Resulting in a more readable code that has very limited clutter and boilerplate code. The API that has been selected to develop the solution is python-sc2 (Dentosal,2018) introduced during the problem-solving part. It deals with the problem of connecting the controller to the model. This is obtained by asynchronously provide the actions for every agent and then let the API and the game do deal with the time step period of communication. This simplifies a lot the project since the timings can be disregarded since there in direct way of controlling time period.

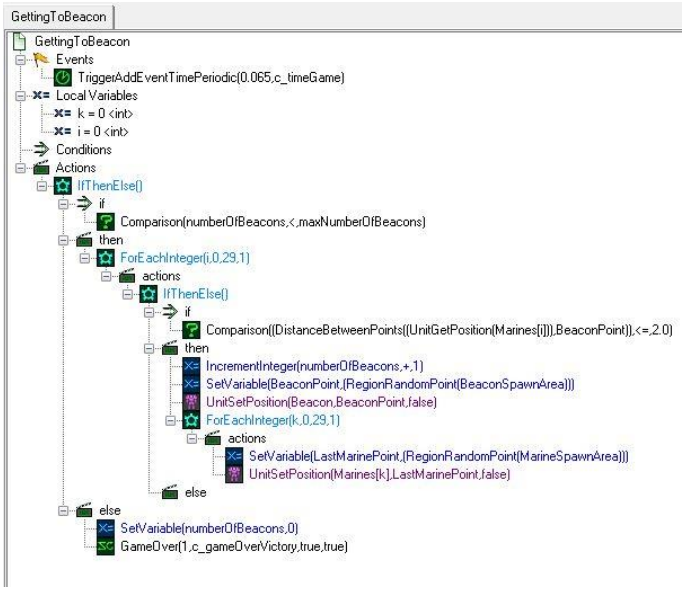
5.1.2 Limitations and Obstacles

The bad intuition was an important initial insight that has been expressed previously in the earlier stages of the project. There were a numerous number of obstacles and limitation that had been faced through the solution implementation. This led to reconsider specific objectives or even simplify objectives so as to be achievable. This was a combination of both the game mechanics and rules along with the capabilities of the API. The API has very specific peculiarities such as running optimally or without errors only in a specific game version. The specific game version can only be found in a peer to peer protocol distribution network. In this case a discord community surrounding the AI in StarCraft 2 provided a magnetic link for version 4.8.4 of the game. Next, the initial assembly of all the tools and resources is not straight forward and one has to be really careful at the beginning since with even minor errors the whole algorithm will not execute but only produce errors.

The game itself is a closed environment. This meant that during the runtime of the scenario there was no way to directly debug the algorithm or even to monitor variables. The input and output information to the game was strictly sieved by the official API in lower layers of the API. This restricts the interactive capabilities of finding mistakes in the code. Furthermore, all errors end in an aborting crashing state. In some other scenarios the game could not deal with the amount of input that was given which caused it to crash unexpectedly. This occurred in scenarios where the algorithm sends information when the game itself is about to terminate. This subsequently caused an error in the API that ends in crashing the program. This indicates that the whole system was easily prone to crashes.
Furthermore, the time and speed variable from the game are not available to the algorithm. Both variables must be treated as constants due to the lock environment. The algorithm does not have control over speed or even acceleration. The only possible move that can be done is to send a movement action towards a specific point position. What is more every action made is not executed by all the agents in a time step but rather every agent individually which also affects the algorithm proposed (E. Ferrante et al ,2016). Another limitation that was unavoidable was the creation of the scenario and the map characteristics using Blizzards game editor. The editor released back in 2010 along with the game. The pseudo programming language used for the editor had a hard learning curve.

5.3 Scenarios Implementation





The above figures show a simple map for a scenario along with the code needed to execute the scenario. The first picture shows the area where the marines' and beacons' position can be selected. The marine is the unit that has been picked as the fundamental agent. The beacon is a round area in the map which represents the goal position. The position is randomly selected every time a new beacon or marine is created. The second figure shows a snippet of the actual code that is needed for the scenario to execute. Specifically, this snippet is responsible for checking whether the flock has arrived at the goal position(beacon). In case this is true the scenario resets meaning that all the marines are moved back to a random marine area and a new beacon position is picked. This iterates 3 times and then the game finishes. Another part of the scenario constitutes the initialisation of

the environment which sets a specific time frame that the game finishes. It also does the initialisation of the variables needed for the beacons and the array of the flock.

5.4 Algorithm Implementation

All the code is available in the git repository added in the appendix, with certain naming conventions that is mentioned in the following parts. Further proof is available in the experimentation stage.

5.5 Collective Behaviour

The whole algorithm has been written in Python. Object orientation was the main architectural decision that was made given the problem along with procedural programming. The API itself promotes the us of object orientation given that the game's agent representation is based on a unit class as defined by the library. One of the strict peculiarities that the API has as a requirement is to have its specific type of unit class. This class has been encapsulated in the marine class which is available in the code. This class is the agent class defined while designing the solution. This class have fields/attributes that indicate the position, if it is informed, the target's position, the encapsulated game marine and its vectors. For the informed marines there is a function that finds the closest target step that lead to the goal in the long run. Another important class implemented is the utility class that implements the mathematical aspects of the algorithm. This class has aspect of procedural programming since the whole class does not store any type of data. Its sole purpose is to act as an application of the mathematical aspects of the algorithm. The main class is called pythonSc2Flock which has the main implemented algorithm as described in the design solution with all the needed parameters, main loop and steps.

5.6Naïve Behaviour

This was an initial implementation that mainly focused on checking that the movement and the calculations where correct. The agents are moving randomly in all possible directions to without any advanced vector calculation nor any informed marines. This was the fundamental step that had to be accomplished to view all the controls and capabilities of the marines. This part is mainly to observe how the movement is executed. Another peculiarity of the game is the how to control the agents. The program has to emulate the movement that an actual player does while playing the game. The program has to select the agent and select the destination. It is important to remember that the acceleration and speed of the unit its static. This mainly affected the magnitude aspect of the vector since which has been changed to one unit, resulting in all agent of the same type to move at the same speed.

5.7 Testing

The methodology that has been used in the majority of the testing endeavours was white box testing. This mean that tests are created that target specifically a class and functionality that its behaviours are known. In other words, there is prior knowledge regarding the implementation of the class. Simple testing classes have been created to check that the utility functionality/procedures are working as expected. Furthermore, simple marine objects have been created to check the functionality of the marine class which is one of the main classes. The most serious limitation was the main class that could not be tested given that the algorithm could only run in one go while the game runs. This promotes inefficient testing due to the real time characteristic of the game that cannot be avoided. That is the reason why testing the collective behaviour and naïve algorithms were not tested. Instead, a different approach has been taken to verify that both acted accordingly. A simple debugging technique has been used to output both in the terminal as well as in a text file aspect of the algorithms while running in real time. This allowed to spot any type of hidden mistakes that were not visible by just observing the simulation.

6. Project Management

To begin with the project overall was a step towards the unknown as mentioned in previous chapters no prior attempts have been found regarding swarm intelligence in StarCraft 2 or its predecessor. This was an alarming factor since I personally had an intuition that I would face a lot of issues and limitations. This resulted in creating a rigorous mentality in tackling the software development part of the project in a weekly basis.

Everything started by setting up every meeting with specific questions/problems, work in progress and achieved functionality. Of course, the first meetings were mainly to familiarise with the problem, then produce questions and further investigate. The next phase was the creation of a project proposal to identify exactly the aim and objectives of the project. Additionally, the aims and objectives were mainly then filtered given the specified time frame available. A Gantt chart was created to indicate a time frame for all the sub-parts of the project such as algorithm development, experimentation and report writing. A contingency plan was formed with the minimum viable project. This plan was precautionary due to the initial intuition regarding the project itself.

The next phase is trying a variety of technologies that are provided that grant access to develop an algorithm for the game. This process was mainly in a trial and error format given that no certain software technology was built for StarCraft 2 based on collective behaviour. The meetings were taking place on weekly basis with my supervisor in a group manner. All the students of the group have projects that were somewhat similar to some degree. This provided a valuable insight to my project since I could discuss my project mainly with my supervisor and secondly with my course mates. This allowed to exchange ideas and arguments during discussions which helped during the development part of the project. During the development phase a weekly journal was recorded with work that occurred on that week sorted by date. The process was mainly focusing on tackling the different objectives individually. Objectives can be altered, paused or revisited depending on the progress made on the objective or the issues that arose. Considering the whole project, I believe that time has been effectively used to develop as well as manage the project.

Communication was a vital aspect of the project since misunderstandings can be easily made regarding certain specifications and objectives. The main gateway for communication was the weekly skype calls that I have attended at the computer science department. A complimentary mean of communication was also a texting service (WhatsApp). This is the application used to send proof of progress made through the week. Additionally, it has been used to set up meetings or inform of effectively the supervisor for any issues. It was a more relaxed informal way of discussing the project which helped to express freely our thoughts. This service also promoted prompt responses regarding matters and a back and forth discussion could be easily formed. During a period, email communication was also suggested by the supervisor on the later stages of the project. The emails were more formal due to the nature of medium used. I personally believe that my supervisor provided all the needed information as well as guidance for the project.

7. Proof and Evaluation

This part of the report includes a plethora of scenarios that were conducted from an experimental point of view. For every scenario a reference is made to all the video resources that are available in the GitLab repository. (see Appendix). This part aims to mainly show the different types of collective behaviour patterns that have been achieved. Additionally, it aims to discuss the parameters that have been identified and implemented along with the turning process. At the end it evaluates the entire process of the project, its outcomes and the content.

7.1 The parameters

The tuning process of the following variables for every scenario was a tedious problem that has been achieved through trial and error. It is not documented in the project due to being outside the primary scope. Instead, proof of the collective behaviour for every scenario is provided along with analysis

- Population size indicates the number of agents in the flock this changes accordingly to every scenario
- Number of informed indicates the number of agents in the flock that know the next step towards the goal
- Social and Individual indicate the weight that is used when calculating the vector of the focal marine. Together the weights should add up to 1.
- Minimum closest number of neighbour parameter is the mechanism that is triggered when the marine has below the set value of neighbours within a certain range.
- Number of neighbours indicate the amount of neighbours that are considered when the focal marine is stranded which then helps the marine to navigate towards them.
- Elasticity is a weight that is between a value from 0 to 1 that indicates how much the vector of every neighbour must considered when calculating the focal marine's vector
- Desired distance is responsible for maintaining a distance between the focal marine and its neighbours. This is enforced for every marine in the flock.
- Neighbourhood radius is the parameter that is used when applying the functionality of finding neighbours within a range. The range corresponds to the radius.

7.2 Proof

7.2.1 Execution of naïve algorithm

Source:(Video 1 "7RandomMarines.mp4")

This is the first step as mentioned previously in an environment with seven marines which correspond to the flock along with a beacon(goal). The behaviour that is exhibited in the video is clear, the marines are moving randomly at any direction. This was an initial step to experiment with controlling the marines. The video describes clearly the fact that the algorithm selects each marine and then does a move action to a selected position. This shows the marine to mindlessly move around the spawning area.

7.2.2 Execution of direct versus collective behaviour

Source:(Video 2 "7MarinesDirectlyToBeacon" and 3"marine7InFormation")

The next step is to create a direct movement of the marines towards the goal. Again, the population size remains at seven. This size has been decided because it can create a formation where six marines are around, and one is in the middle. All the marines move directly towards the goal since the know the position of the beacon. This is can be seen from

the video 2 where the marines without having any aspects of flocking move towards the goal. One can imagine that the marines move on their own disregarding their neighbours. Video 3 shows the marines moving towards the goal but also maintaining the formation created. One of the marines is the informed marine which knows the next step towards the goal. This marine can be easily spotted since it acts as a leader to the flock. This information is indirectly passed through from neighbourhood to neighbourhood. The important aspects are the factor that the marine move within a formation as well as the fact that no jerking motion occurs, nor the flock stays stationary to a location. This is the best-case scenario that can be observed especially in small flocks. This is the correct collective behaviour that has been extracted from the robot flocking paper (E. Ferrante ,2016)

7.2.3 Execution of collective behaviour in larger flocks

Source:(Video 4 "20Marines2Informed")

The next stage is to increase the number of marines since flocks of animals tend to be in larger numbers. The population size has been increased to 20 and the number of informed marines to 2. This creates 2 marines that act again as leaders to pull the entire flock. At the beginning of the scenario the marines are scattered. This triggers the system of the stranded marine that makes all the marine to regroup. Next, the collective behaviour is applied to create the formation and move towards the goal. Another interesting characteristic is the fact that the formation is totally different in this situation and the group have a different spread. In some situations, depending on the movement the leader may stand out from the formation due to a stronger pull towards the individual goal vector compared to the social vector.

7.2.4 Execution with issues

Source:(Video 4 "InformedPullingTheSwarm")

This is a scenario where the collective behaviour has not worked. This is what exactly is caused when the parameters of the algorithms are not properly tuned.In the video 4 the population increased to 30 with 2 informed marines. The first characteristic that seems to be not applied is the spread of the flock. The marines seem to aggregate towards the middle of the flock. Furthermore, it seems that the marines literally are pushing out of the way each other. This leads to neighbours which are shifted around in different neighbourhoods. This creates a motion that is moving diagonally. The flock also can be observed moving in a circular motion due to the neighbourhood alterations. This subsequently causes the group to move in a slower pace. The informed marine tries to pull the group but to no avail, causing it to move in smaller increments. Sometimes it also distances from the group enough which triggers the stranded mechanism that guides back to the group. This is displayed by the marine going back and forward.

7.3 Discussion

First and foremost, the step of tuning the parameters is very important. This is primarily evident from video 4. The lack of tuned parameters cannot produce any good type of behaviour besides the chaotic diagonal movement described previously. Once the parameters are tuned, then good collective behaviour emerges. The most important characteristic seen by the collective behaviour is the spread. This mainly relies on the population size along with the desired distance. Overall, it has been observed that larger flocks tend to have a larger spread. This is applicable in the reverse order where smaller flocks have a smaller spread. The second interesting characteristic is the formation created by the flock. In the case of the 7 agents in a group an arrow or a circle like shape was created. In larger number the circle overpowered any other shape, in groups of 20 it has been noted a circle within a circle formation. This leads to the conclusion that there are a few shapes still hidden that have yet to be discovered. Furthermore, it has been realised that the stranded mechanism did not negatively affect the collective behaviour. Quite the contrary it seems like an effective way to regroup the population so as to perform the behaviour.

7.4 Evaluation

This part dives in the realisations that occurred through the life span of the project. Numerous lessons have been learnt which will act as valuable experience for the future. The most distinct bittersweet experience is the issues, obstacles and limitations that have been faced. This lead to change the plan that was initially discussed as well as compromising some of the objectives and outcomes of the project. The reason was mainly the exploration to unknown territory. At the very beginning, it seemed so hard to discharge of any problems. This was faced when I started using the APIs, in a case I developed and experimented with an API more that I should. Even though I could not find what I expected only to arrive at a dead end (technical issue). Even the selected API had some limitations that may discourage in further investigation due to the current state of library and the game. It is complicated enough to set up the game with the API which is one of the first required steps to be able to even consider undertaking such a project. I should have limited the time spend considering libraries and be more open to alternative solutions. The findings that occurred through the whole project were quite unexpected and hard to perceive. Especially the formation of the flocks and their physical spread. The expected behaviour which has been described in the early stages has been successfully implemented. All the functionality required has been accomplished and displayed. Experience has been gained and intuition for parameter tuning for the StarCraft 2 environment. Overall, the project had both its bad and good moments that are invaluable lessons for the future.

8 Conclusion

8.1 Future work

To begin with the API used needs additional support so as any version of the game is usable. This is one of the fundamental steps that shall occur so as to have a smaller number of errors. This project emphasised only to a specific ground unit called the marines. This is evident from the experiments conducted. The use of other units can lead to new understandings of the game as well as new parameter tuning experience. The environment can be changed by introducing more buildings and enemy units so as to observe the resulting behaviour. A different aspect can be introduced along with the collective behaviour such as reinforcement learning. For this a new set of scenarios must be created having in mind the reinforcement learning paradigm. A very important characteristic is the introduction of reward and penalty that can be based initially to simple immediate strategies that the flock can learn. For instance, the amount of time reaching a position that is closer to the next step towards the goal. Another example is reaching the goal with minimal casualties, this is based assuming that enemy units or building can harm the flock. Next these two characteristics can be combined in a sense that the flock will try to find a balance between maximising the time reaching the goal and the amount of damage that has been received. Furthermore, additional GUI (graphical user interface) feature can be developed to show in real time in a different window the movement vectors of every agent and the overall movement vector of the flock. In addition, all the agents can be represented as a triangle with an orientation and different colours. The representations should change colour when depending on the state of the agent (fighting, moving, informed and focal). This could provide a direct set of information when observing the behaviour. A really long process of the project was to fine tune the parameters of the algorithm this differ based on the flock size and the units. A possible solution is to develop an evolutionary algorithm that aims to find a good set for the parameters

8.2 Conclusion

Overall the aim of this project is to explore a new path that was never attempted previously. Introducing the flocking behaviour to StarCraft 2 was successfully achieved. Settings and refining the parameters for the algorithm seemed as a tedious process that finally produced the expected behaviour. As mentioned during the literature review this project is primarily inspired by a robotics collective behaviour. The transferability and refinement of the algorithm has been achieved even with all the obstacles and limitations that have been faced due to the nature of the game. The project specifications have been compromised and altered to conform to the limitations. The contingency plan worked as stated in the project proposal. This was quite a unique experience with such a contemporary project. Once this first steps have been achieved, it is important to continue and develop further the project in future project iterations. To sum up, the project is a successful introduction of swarm intelligence in a new territory of StarCraft 2 which may have great potential.

Appendix

References

Pinciroli, C., Trianni, V., O'Grady, R. et al. Swarm Intell, ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems (2012) 6: 271. https://doi.org/10.1007/s11721-012-0072-5.

Stracraft 1 (first edition). 1998.PC[Game]. Blizzard Entertainment: California.

Stracraft 2 (second edition). 2010.PC[Game]. Blizzard Entertainment: California.

Stracraft 2 Editor (second edition). 2010.PC[Game]. Blizzard Entertainment: California.

C. Reynolds. Flocks, herds and schools: A distributed behavioral model. In M. C. Stone, editor, SIGGRAPH '87: Proc. of the 14th annual conference on computer graphics and interactive techniques, pages 25–34, New York, 1987. ACM Press.

I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin. Effective leadership and decisionmaking
in animal groups on the move. Nature, 433:513–516, 2005. doi: 10.1038/nature03236.

S. J. Simpson, G. A. Sword, P. D. Lorch, and I. D. Couzin. Cannibal crickets on a forced march for protein and salt. Proceedings of the National Academy of Sciences, USA, 103 (11):4152–4156, 2006.

Eliseo Ferrante, Ali Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, et al.. Self-organized flocking with a mobile robot swarm: a novel motion control method. Adaptive Behavior, SAGE Publications, 2012, 20 (6), pp.460-477. 10.1177/1059712312462248.

Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, Marco Dorigo. Swarm Intell Swarm robotics:a review from the swarm engineering perspective. Swarm Intelligence, Springer, 2013, 7 (1), pp.1-41.10.1007/s11721-012-0075-2.

Blizzard (2017) Sc2client-api (Version 1.1.0) [API]. Available at: https://github.com/Blizzard/s2client-api (Last Accessed:05/09/2019).

Dentosal (2018) python-sc2 (Version 0.11.1) [API]. Available at: https://github.com /Dentosal/python-sc2 (Last Accessed:05/09/2019).

DeepMind (2017) pysc2 (Version 2.0.2) [API]. Available at: https://github.com/deepmind/pysc2 (Last Accessed:05/09/2019).

Node-Sc2 (2018) core (Version 0.8.2) [API]. Available at: https://github.com/node-sc2/core (Last Accessed:05/09/2019).

ocraft (2019) ocraft-sc2client (Version 0.3.9) [API]. Available at https://github.com/ocraft/ocraft-s2client (Last Accessed:05/09/2019).

NikEyX (2018) SC2-CSharpe-Staterkit (Version 0.0.1) [API]. Available at https://github.com/NikEyX/SC2-CSharpe-Starterkit (Last Accessed:05/09/2019).

Oriol Vinyals Timo Ewalds Sergey Bartunov Petko Georgiev Alexander Sasha Vezhnevets Michelle Yeo Alireza Makhzani Heinrich Kuttler ¨ John Agapiou Julian Schrittwieser John Quan Stephen Gaffney Stig Petersen Karen Simonyan Tom Schaul Hado van Hasselt David Silver Timothy Lillicrap.(2017) 'StarCraft II: A New Challenge for Reinforcement Learning'.DeepMind.

Code

Git Repository: https://git.cs.bham.ac.uk/sxl1028/sc2-swarm-intelligence

Access is provided upon request to this private repository. The supervisor has access to all the code from the start of the project. All the code has been developed by me except the API that has been used python-sc2(Dentosal,2018). Furthermore, there is a folder called "ExperimentsAndProof" that include the proofs that the collective behaviour occurs. These are the video proofs that were provided during the presentation stage of the project.

Setting up Project

Windows

- Install pip3 and python3 for windows.
- "pip3 install –user –upgrade sc2" to download the library.
- Download the 4.8.4 version of StarCraft 2.
- Navigate to the game directory and add the executeInfo.txt in the StarCraft 2 folder
- Create a sub folder called Maps in case is not present and inside create another subfolder called "mini_games", add all the .SC2Map files in the "mini_games" subfolder you created.
- "pip show sc2" this shows the directory where the API is installed.
- Navigate to directory and find the file sc2/paths.py and edit this file
- In the BASEDIR on the "Windows" parameter you should insert/replace the correct directory of the game
- In the USERPATH you have to edit the "Windows" parameter to point to the ExecuteInfo.txt file you previously added.

Execution

- Navigate to the directory of the "pythonSc2Flock.py" file
- Run the command "python pythonSc2Flock.py"
- This will launch the game in a scenario and the collective behaviour should be observable

Git Log Commits

commit 1eb8b9e0dc2a6f5b2a528efd13824450f4bea7b2
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Mon Sep 9 14:59:14 2019 +0100

Including all the information needed and proof videos of the achieved behaviour

commit a2790abd8abfa8813cc75aad42902efb5d498a30
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Wed Aug 7 16:42:11 2019 +0100

7 Marine running in formation works

commit f6d03a76129c1c0d30d512a164e20d364fc8f04d
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Tue Jul 23 16:20:05 2019 +0100

change closest to radius based trying to stabalise

commit 51d29de49f2392a1326440fa52c65fa528bc9fb4
Author: sxl1028 <sxl1028@student.bham.ac.uk>

Date: Mon Jul 22 01:53:08 2019 +0100

Added social and individual parameters, fixed addition of y component was wrongly doing x started using coordinates class

commit 0e84dc6ace6456481437885b6bbcfad3d643331f
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Tue Jul 16 16:05:43 2019 +0100

Trying to add focal,refactoring code

commit 10cc016a859a27fa963847f3b94230767778f086
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Mon Jul 15 22:27:39 2019 +0100

Trying to implement pseudocode given

commit 2549695de7b1dfae6e0406f065c3e5150585e86c
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Wed Jul 10 12:37:02 2019 +0100

Adding multiple vectors and tested it

commit faa48b8b164af63718624e30c3601ca6c323798d
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Wed Jul 10 12:12:06 2019 +0100

Adding calculations of finding the closest marines of a marine

commit 9aa25220f8c70a5c3cd994cbfefdace3a407c47f
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Tue Jul 9 19:54:26 2019 +0100

Change code refactoring math utils to separate class and create fundemental classes

commit 22432c45e6e5efe1a95b4c57574fe8d56481cb5b
Author: sxl1028 <sxl1028@student.bham.ac.uk>
Date: Mon Jul 8 11:49:59 2019 +0100

Finally creating a git to track progress of this project

commit 4cde6b7bb1b9f0f0dfdc7d780407d6601f2bcea5
Author: sxl1028 <sxl1028@cs.bham.ac.uk>
Date: Mon Jul 8 11:21:29 2019 +0100

Initial commit