## Group robotic behaviour simulation using Robot Operating System (ROS)

## Project Team Name:

## Oriental Turtles

## Project Team:

**Amruth Akoju**

UPI: aako822

**Jahnavi Bharadwaj**

UPI: jbha386

**Charindu Hashan**

UPI: twee009

**Sunkee Hong**

UPI: shon065

**Kyungmo (Peter) Kang**

UPI: khan059

**Santhan Kusam**

UPI: skus887

**Ryun Madenokoji**

UPI: rmad019

**Ken Neth Yeoh**

UPI: kyeo475

## Project Client:

**Bruce MacDonald**

Email: b.macdonald@auckland.ac.nz

# Objectives

The aim of the project is to implement and coordinate individual robotic nodes to achieve a collective behaviour, using Swarm Robotics. These robots will be designed in such a way that they indicate group behaviour by sending messages to each other. The robot behaviour simulation will be carried out using ROS (Robot Operating System). The main components to be included are: farm area, grass, sheep, soil, water and farmers. The objective is to create these robots and implement various agricultural properties for them. The behaviours of the robots will be either autonomous, or controlled by users.

# Scope

The project objective is to design and graphically simulate an example of Swarm Robotics, with the requirements set by the client, within six weeks.

1. **Functional requirements:**

Specify functions that a system or component must be able to perform.

* Produce robot nodes which represent Farmer, Sheep and Grass.
* Produce a farm which contains at least four rectangular fields.
* Grass that can grow and decay.
* Sheep that can move, eat grass, grow, excrete, sense grass, get ill and recover.
* Patches of soil with different qualities.
* Random distribution of rain across all fields.
* Farmer that gathers sheep and moves them to a truck, to sell them for money.

1. **Code requirements**

An estimate as to how the code will be assembled to accomplish the functional requirements.

* Use C++ to code.
* Use ROS (Robotic Operating System) to control robotic nodes.
* Grass class is dependent on water, soil, and sunlight. Use a simple formula to calculate growth and decay rate.
* Include a parameter called ‘soil’ for each patch of field, which determines the quality of the soil.
* ‘Farmer’ class used to gather sheep and move them to the truck.

1. **Non-functional requirements**

* The wiki and repositories on GitHub should be available for client’s viewing at all times.
* Backup of the project should be available on university drives and GitHub
* The repositories on GitHub should be made accessible only to team members and the client.

# Assumptions

* Members of the project will be available for the entire project duration.
* Client will be available to discuss project queries during the entire project.
* The members of the project have sufficient knowledge and skills required to complete the project.
* Project team members will adhere to the communications plan.
* There is no additional cost involved in this project.

# Constraints

* The team members have no prior experience in C++ and ROS.
* ROS only supports Ubuntu as an Operating System. The others are still in experimental phase.
* The project has limited time frame; the project plan needs to be presented within a short fixed time.

# Approach and Methodology

Waterfall methodology will be used for this project. It is a linear and sequential approach to software design and development. Within each waterfall stage, tasks are assigned to dependant subgroups to ensure greater project and deadline control. It has five stages: Requirements, Design, Implementation, Verification/Testing, and Maintenance.

The justification for using this methodology is the time constraints the project has. The requirements for this project has been clearly set at the initial stage by the client, thus it would be more constructive to undertake the project with this methodology. It allows for thorough planning, especially for logical design implementation and deployment.

# Communication

Team meetings are to be held at least twice a week. The frequency and lengths of the meetings will be adjusted during the different stages of the project.

Each meeting will have an agenda and reports from individuals on the things that have been accomplished since the previous meeting. One member will be responsible to take down the minutes and upload them onto GitHub after each meeting.

Communication within the team and with the client will be carried out in various ways:

1. **Facebook**

It is a medium which team members can use to communicate.

* Communication within the team via Facebook Group ( *[SE 306] Group 3*).
* Communication with the client via Facebook Group ( *[SE 306] Q&A with lecturer*).
* Sharing of unofficial documents.
* Discussion platform for addressing meeting agendas and reminders.

1. **GitHub**

It is a medium where the team can upload official documents. The client has access to this in order to follow the progress of the project.

* Repositories
* Wiki
* Meeting agenda and meeting minutes

1. **Scheduled Meetings**

* Meetings are to be held at available venues around the University of Auckland campus.
* During the initiating and planning stages, team members are encouraged to ask the client any questions about the requirements.
* During the initial stage, meetings are to be held around a meeting table to allow easy face-to-face communication between team members.
* During the implementation stage, meetings should be held in labs, with computers, to allow demonstration of what has been accomplished.

# Workload Allocation

For the programming sections, it was decided that the best way to accomplish the requirements of the project was in pairs, while maintaining constant communication between all members of the group. This can be a very efficient way of working, as it ensures that every member of the group will be able to code. They will always have support from the other members of the group in case they come across any difficulties. All the design decisions will be made when all the group members are present, and they will be implemented by the allocated pair. Pair programming can also be an effective way to filter any mistakes or errors that one member may commit while programming. The pairs were allocated as follows:

**Pair One:** Amruth, Ken Neth

**Pair Two:** Charindu, Peter

**Pair Three:** Jahnavi, Santhan

**Pair Four:** Ryun, Sun Kee

Each pair will be allocated to a specific component, which they will be responsible for creating, and implementing with the required properties.

Allocations for implementation:

1. **Make components**

* Grass, Farm (4 fields) Pair Four
* Sheep Pair One
* Soil (One type for each field), Water Pair Three
* Farmers Pair Two

1. **Properties of each component**

* Grass (dies, grows), Farm (4 fields) Pair Four
* Sheep (eats, grows, exercises, has illnesses, excretes, senses grass) Pair One
* Soil (have different qualities for each field),

Water (random falling speed for each field) Pair Three

* Farmer (herds, moves, sells sheep) Pair Two

1. **Movement of components**

* Grass (growth speed) Pair Four
* Sheep (moving, growing, excretion) Pair One
* Water (falls down with speed) Pair Three
* Farmer (handles sheep) Pair Two

The **testing** for each of the above three tasks will be primarily carried out by Pair Three, with other members of the group making changes, if necessary.

Pair Two will mainly be in charge of the **documentation**.

Pair One will be focussing on the **refactoring**.

# Testing Plan

Throughout the development phase of this project, we will be conducting regular tests to verify that our system is working as it should. After a programmer has finished developing a feature, he/she will conduct thorough unit tests to verify that the system is behaving as it should. Test cases used for unit testing will be written by the programmer as they are writing code for the main system.

Once the system has passed all the unit tests, it will then be passed on to the merging/testing team. They will merge the new features with the master system (stable release) and perform integration testing to check if the new features integrate into the master system without causing other features of the system to malfunction. After it is confirmed that the new features are stable and not interacting negatively with other parts of the system, they are then merged to the master branch. If these tests fail, the merging/testing team will work closely with the programmer responsible for this feature to rectify the issues.

After the new feature is merged into the master branch, the merging/testing team will perform more tests to validate that the product that is being released meets the specifications laid out in the project brief. When the testing team and the rest of the project members agree that the system is fit for release, it will then be moved into the release branch of the Git repository.