UNIVERSITY OF PLYMOUTH

# Ant Behavior Simulation Project

Najim Mazidi

**Student ID: 10438524**

**1/14/2016**

# Table of Contents

# INTRODUCTION

This report demonstrates a program that simulates the movements, actions and communication of a number of *n* ants and *m* robber ants. The program is built to mimic the actions of ants in real life in order to help the user gain a better understanding of the way ants forage for food. The program is written in C# and developed in Microsoft Visual Studio 2012.

The main objective of both the ants and robber ants is to look for food and when found, go back to their nest and deposit the food. As an individual, an ant is merely powerless and will not be able to survive. However, when there are hundreds, thousands or even millions of ants in a group, they can collectively carry out a number of very complex tasks in order to survive. This is known as collective intelligence. The program described in this report will try to emulate this.

# DESCRIPTION OF SOLUTION

## Classes used:

- AntColonyForm
- Ant
- Nest
- Food
- RobberAnt
- RobberNest

Shown above is a list of all the classes used in the program. The AntColonyForm class is the main class that contains the form design and all the main code in the program. As the program is coded in an object-oriented manner, there are several additional classes for the ants, nests, food, robber ants and robber nests. The purpose of these classes is to store information about each ant, food, etc. such as if an ant is carrying food or not. There are lists, which will be explained in more detail further into the report, that are linked to these classes. Also, these classes are used to create each ant or food.

Below are the unified modelling language (UML) diagrams of each class in the program and a description and annotation of each.

## ANTCOLONYFORM

The AntColonyForm class consists of a number of fields and methods. antList is a list of 150 ants, which are created when the form is loaded. Lists allow the program to store a non-fixed number of values. In this case these values are ants, which are created to collect food and store it into their nest. The antList stores each ant in an index of 0 to 149. An ant is created by simply using *tempList = new Ant,* and then adding tempList into the list by using *antList.Add(tempList)*. Each ant can then be accessed by using the *ElementAt()* method.

foodList is similar to antList as it stores all the food sources. The user can add a new value to the list by left-clicking on a specific location. The x and y coordinates are stored in the Food class and list.

nestList stores each ant nest that is created. The user can do this by right-clicking on where they want the nest to be created. Similar to foodList, the x and y coordinates are stored in the Nest class as each nest is created.

robberNestList and robberAntList do the same as nestList and antList, respectively. The only difference is that they store the 'robber' ants and their nests.

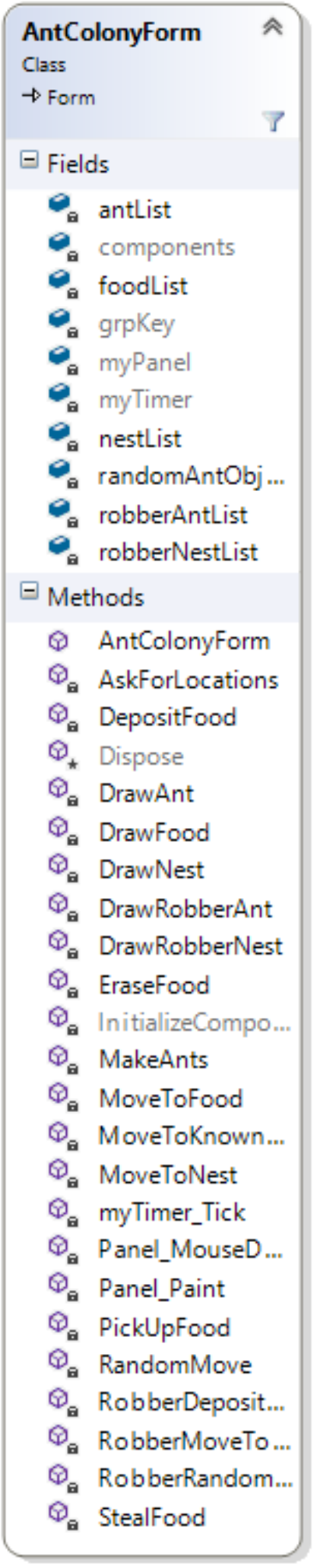
randomAntObject is used to generate a random number. This is used when creating the ants. The random numbers are the x and y coordinate of each ant. This way, every ant is in a random position when the form is loaded.

The AntColonyForm method is the default starting method. This means that when the form is loaded, the code starts from this method.

myTimer_Tick is an event handler that runs the code within it every time that the timer ticks. In this case, when the timer clicks, the ants move in a specified direction.

The MakeAnts method is used to initialise and create every ant and store it in the list. This creates both the ants and the robber ants. For every instance if the loop, a new ant is created and stored with random coordinates.

The DrawAnt method, when called, uses c# graphics to draw the ant as a rectangle on the panel and wipes it's previous location in order to move it. When calling this method, the previous x and y coordinates and the index of the ant have to be provided as properties. The DrawFood, DrawNest, DrawRobberAnt and DrawRobberNest methods work in the same way.



When an ant comes close to another ant, it can communicate with it by asking for the locations of the nest or food source. If the ant doesn't know the location of, for example the nest, and the ant close to it does, it can learn the location of the nest from it.

DepositFood is called when an ant is close to a nest. It then no longer is carrying any food so it moves back towards the food source or moves randomly until it finds more food. For the robber ants, RobberDepositFood is called.

Each food source holds 500 units of food. When the ants collect all 500 units, the EraseFood method is called to erase the yellow ellipse from the panel and remove the food source from the list.

If the ant knows the location of the food source, the MoveToFood method is called. If the x coordinate of the ant is less than the x coordinate of the food, the ant's x coordinate is increased by one, and vice versa. MoveToNest works in the same way, it just moves the ant closer to the nest instead of food.

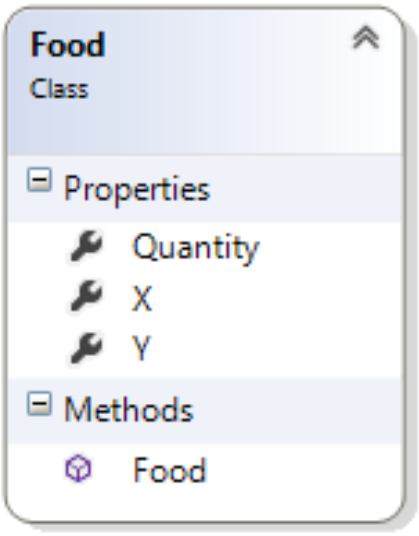
PickUpFood is called when the ant is close to a food source. I makes the ant pick up a unit of food so it sets the *isCarrying* property of the ant to true.

When the ants are created, they do not know the location of the food or nest so they move randomly until they come across a food source or a nest. The RandomMove method is used to move the ant in a random direction. There are eight possible directions for the ant to move so a random number between one and eight is generated. For robber ants, RobberRandomMove is called.

StealFood is called when a robber ant comes across an ant. The robber ant takes the food from the ant and moves towards the robber nest.

## FOOD

The Food class holds information about each food-source that is created by the user. These food-sources are stored in the food list. The X property holds the x-coordinate value of the food-source and the Y property holds the y-coordinate. These coordinates are set when the user clicks on a position in the panel. So for example, if the user makes a left mouse click at coordinates (50,100) the food will be create at those coordinates and the values if X will be 50 and the value of Y will be 100. The food is then drawn on the panel as an ellipse using a yellow brush.




The Quantity property is used to store the number of units of food that is stored in a specific food source. This number decrements every time an ant picks up food from it.

The Food method allows the class to be called externally and sets the x and y values that are inputted as the X and Y properties.

## ANT


The Ant class holds certain information about the ants in the program. The ants that are stored in antList are of type Ant so they each have the following properties:

The X value holds the x-coordinate of the position of the ant in the panel and Y holds the y-coordinate.

The closestAnt property is an integer value. It holds a value for the index of the closest ant in antList. For example, if ant number 34 is the closest to the ant in question, the value of closestAnt will be 34.

foodClosest is similar to the closestAnt property. Instead of holding the index of the closest ant, it holds the index of the closest food source to the ant in the foodList list.

foodLocationKnown is a Boolean property which holds a true of false value. If the ant in question knows the location of a food source, then the value would be true.

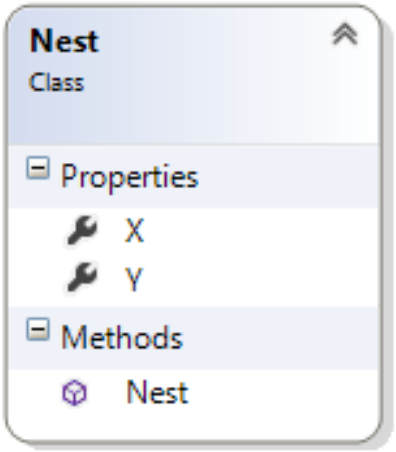
The isCarrying property holds a Boolean value that determines whether the ant is carrying any food or not. If it is carrying, the value would be true, and vice versa.

isCloseToAnt holds a true or false value. If the ant in question is within a radius of r of another ant, the value of the property will be set to true, if not then it would be set to false. isCloseToFood and isCloseToNest are similar. They would hold true if ant is close to a food source and if ant is close to a nest, respectively.

nestClosest and robberNestClosest hold integer values for the index positions of the closest nest to the ant in the nest list and the closest robberNest to the ant in the robber nest list.

The Ant method sets the X and Y properties to random values and logs the creation of the ant in the console.

## NEST




The nest class holds information about each nest that is created by the user. Every nest that is created is stored in the nestList list. The X and Y properties hold the x and y coordinates of the nest. This is the position that the nest exists at in the panel.

## RobberAnt


The RobberAnt class holds information regarding each robber ant that exists in the program. The robber ants are stored in the robberAntList list and are of type RobberAnt so they have the same properties.

The RobberAnt method sets the X and Y properties to random values and logs the creation of the robber ant in the console.

The X and Y properties are the coordinates of the location of the robber ant on the panel. These are random numbers when first created.

This class has some properties that have the same definitions as the Ant class. These properties that are unique to the RobberAnt class are:

lastAntX and lastAntY hold the x and y coordinates of the ant that the robber ant has previously stolen food from. This is used when the ant has deposited food in the nest and wants to go back to the previous positions in order to come across more ants that it can steal food from.

robberNestLocationKnown holds a Boolean value. If the robber ant knows the location of a robber nest, this value would be true, if it doesn't then the value would be false.

# Evaluation

The program works correctly and fulfils every part of the specification to a good standard. Although not included in the original specification, the robber ants were implemented in order to simulate the ant world more effectively and to bring in a new aspect to the program. With this implementation, the ants behave noticeable differently when the robber ants are stealing food from them as their actions and movements become different and less effective thus causing a slower transfer of food from the food-source to the ant nest.

I believe that the coding aspect of this project was carried out exceptionally well. This is due to a number of reasons. One of these reasons is because the coding was well planned prior to carrying out the coding. Another reason is because object oriented programming (OOP) was used when writing the program and the high number of different classes. This made the program easier to understand and made it run more efficiently thus avoiding the being 'spaghetti code'.

One aspect of the code that sometimes does not work entirely correctly is the *AskForLocations* method in the *AntColonyForm* class. If the ants are moving quickly while

moving to and from the nest, if they go past another ant they sometimes don't share information as they go past it too quickly so there is no time for the timer to tick in order to make the program call the *AskforLocations* method. This is a rare problem and affects the overall performance of the program very slightly. Thus this problem would only be considered as very small, or unnoticeable.

In hindsight, if there was more time to carry out this project, I would have added more features in order to improve the program. One improvement could have been the inclusion of images of the ants, nests and food. So for example, instead of ants being represented as black squares on the screen, there could have been an image of an illustration of a real life ant.