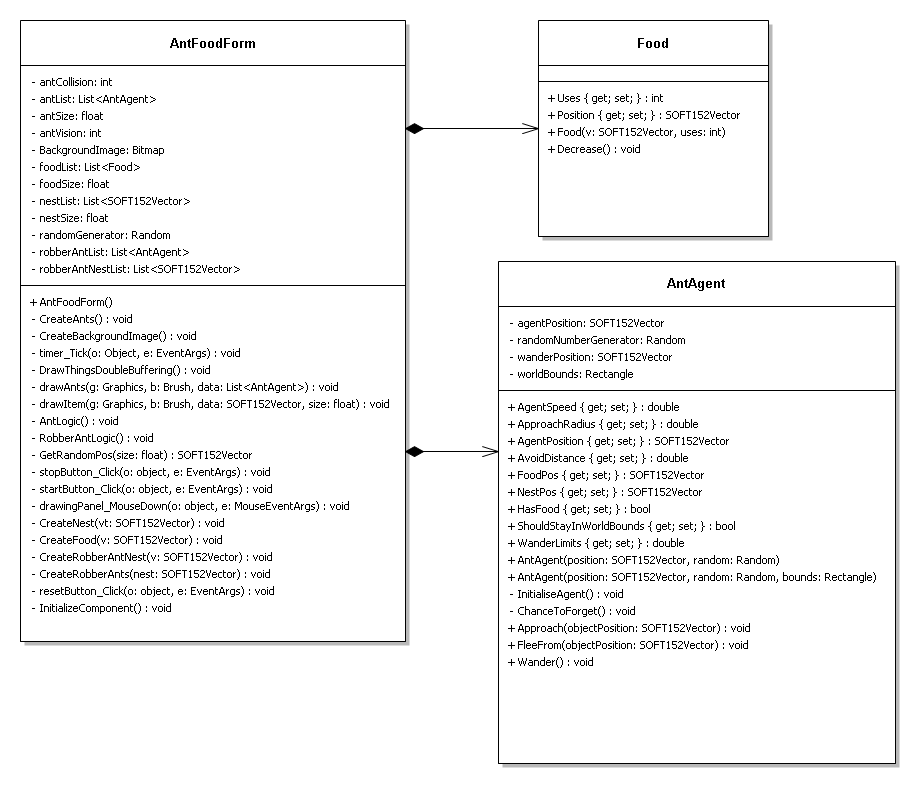
SOFT152 Option 2 Report

# Class Structure

The solution created involved the use of 3 classes, AntAgent, Food and AntFoodForm. Below is the UML class diagram.



Although not included in this diagram the given classes SOFT152Vector and Steering were used, with both Food and AntAgent having composition relationships to SOFT152Vector.

## Food Class

### 

The food class defines an object (food) that the agents interact with in the world to receive food. It has a number of uses and a position in the world defined by the 2 properties.

Module Decrease

Input: none

Output: none

uses := number of uses the food has

uses := uses -1

End Module

## AntAgent Class

### 

The AntAgent class defines a single agent, it has a position (AgentPosition) and a memory defined by the FoodPos and NestPos properties which are the locations of the food and nest respectively. The agent has 2 states it can be in has food or does not have food, this is defined by the HasFood property. The properties approachRadius, avoidDistance, agentSpeed and WanderLimits are all used for the 3 movement methods and control how fast an agent moves and how an agent approaches an object.

//module which gives the ant a chance to forget its nest and food positions

Module ChanceToForget

Input: none

Output: none

nestPosition := the position of a nest

foodPosition := the position of a food

forgetChance := random number between 0 and 100

IF (forgetChance > 96)

IF (forgetChance > 98)

nestPosition := null

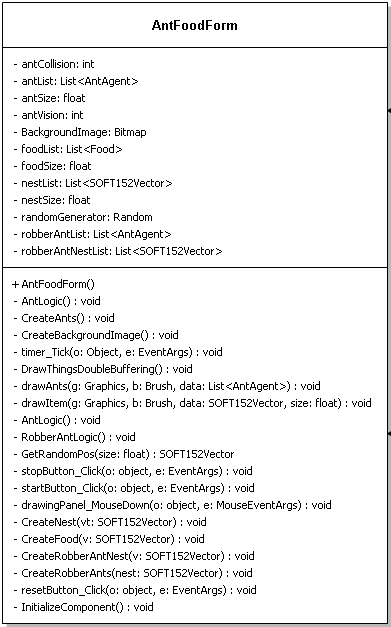
ELSE

foodPosition := null

ENDIF

ENDIF

End Module

This class is the form class that creates the user interface and actually creates instances of the other classes.The fields in the antfoodform are used to store lists of ants foods, and nests as this makes it easy to perform many tasks (such as controlling movement) on a large group of object. And the remaining fields (antCollision, antSize, antVision, foodsize and nestSize) area used to draw the objects onto the bitmap and to control hitboxes of the ants and interaction radiuses.

//module which creates all the ants required

Module CreateAnts

Input: numberOfAnts – the number of ants to be created

Output: AntList – a list of ants created

FOR index FROM 0 TO index < numberOfAnts DO

PositionX := a random number

PositionY := a random number

Create a new ant at positionX and positionY

Add new ant to antList

ENDFOR

This module was altered slightly with the creation of the following module resulting in the postionX and position Y to be replaced with a call of the following module as during test phases this is how food and nests were created.

//get a random valid point in the world

Module GetRandomPos

Input: sizeOfObject- size of the object, worldWidth – width of the world, worldHeight- height of the world

Output: posX , posY- a random position within the world

validRangeX := worldwidth – sizeOfObject

validRangeY := worldheight - sizeOfObject

posX: = random number between 0 and validRangeX

posY := random number between 0 and validRangeY

End Module

//Module which creates robber ants at a given nest

Module CreateRobberAnts

Input: NestPos- the position of the ants to be created at

Output: robberAntList – a list of robber ants

numberOfAnts-number of ants required

FOR index FROM 0 TO index < numberOfAnts DO

antPosition := nest Position

create new ant at antPosition

Add new ant to robberAntList

ENDFOR

End Module

//Module which draws all ants from a list

Module drawAnts

Input: antList – a list of ants

Output: all ants drawn

antListCount := the number of ants in antList

FOR index FROM 0 TO index < antListCount DO

currentAntPos := AntList[index] current position

draw ant at currentAntPos

ENDFOR

End module

//module which controls ant interactions and movement logic

Module AntLogic

Input: Ant – an ant, nestList- a list of all nests, foodList- a list of all foods, antList- a list of all ants

antPos := the ant current position

mNestPos := ants memory of nest position

mFoodPos := ant memory of food position

antVision := the distance an ant can see

antCollision := the collision size of an ant

hasFood := Boolean that states whether ant has food or not

nestListCount := number of nests in the list

foodListCount := number of foods in the list

antListCount := number of ants in the list

FOR index FROM 0 TO index <nestListCount DO

nestPos := nest position

IF( distance between antPos and nestPos < antVision)

mNestPos := nestPos

ENDIF

IF( distance between antPos and nestPos < antCollision)

Ant move away from nest

IF(hasFood = true)

hasFood := false

ENDIF

ENDIF

ENDFOR

FOR index FROM 0 TO index < antListCount DO

otherAntPos := an ant in the list position

distance := distance between antpos and OtherAntPos

IF( distance < antVision)

Share information between both ants

ENDIF

If( distance < ant Collision)

Ant move away from otherAntPos

ENDIF

ENDFOR

FOR index FROM 0 TO index < foodListCount DO

foodPos := food position

FoodUses := number of uses a food has

distance := distance between antPos and FoodPos

if( distance < antVision)

mFoodPos = foodPos

ENDIF

IF(distance < antCollision)

Ant move away from foodPos

IF (hasFood == false)

hasFood = true

Slow Ant

Decrease food Uses

ENDIF

IF(foodUses = 0)

Remove food at food[index]

ENDIF

ENDIF

ENDFOR

IF(HasFood = true AND mNestPos != null)

Move toward mNestPos

ELSEIF( hasFood = false AND mFoodPos != null)

Move toward food

IF(distance between antPos and mFoodPos < antVision)

mFoodPos := null

ELSE

Wander around randomly

End Module

//Module for controlling robber ant interactions and movement logic

Input: Ant – an ant, nestList- a list of all robber nests, antList- a list of all ants, robberAntList – a list of all robberAnts

antPos := the ant current position

mNestPos := ants memory of nest position

mFoodPos := ant memory of food position

antVision := the distance an ant can see

antCollision := the collision size of an ant

hasFood := Boolean that states whether ant has food or not

nestListCount := number of nests in the list

antListCount := number of ants in the list

robberAntListCount := number of ants in the list

FOR index from 0 TO index < antListCount DO

anotherAntPos := antList[index] position

anotherAntHasFood := status of ant at antList[index]

distance := distance between antpos and anotherAntPos

IF(distance < antVision AND anotherAntHasFood = true)

Ant go to anotherAntPos

anotherAnt run away from Ant

ENDIF

If(distance < antCollision AND anotherAntHasFood = true)

hasFood = true

anotherAntHasFood = false

mFoodPos = anotherAntPos

slow agent

ENDIF

ENDFOR

FOR index FROM 0 TO index < robberAntListCount DO

robberAntpos := position of the robberAnt

distance := distance between robberantpos and antpos

IF(distance < antVision)

Share information

ENDIF

IF(distance < antcollision)

Move away from robberAntPos

ENDIF

ENDFOR

FOR index FROM 0 TO index < nestListCount DO

nestPos := position of current nest

distance := distance between robberantpos and antPos

IF(distance < antvision)

mNestPos = nestPos

ENDIF

IF(distance < antCollision)

IF(hasFood = true)

hasFood = false

speed up agent

ENDIF

ENDIF

ENDFOR

IF(mNestPos != null AND hasFood = true)

Move to mNestPos

ELSIF (mFoodPos != null AND hasFood = false)

Move to mFoodPos

distance := distance between antpos and mfoodpos

If( mFoodPos != null AND distance < antvision)

mFoodPos = null

ENDIF

ELSE

Wander around randomly

ENDIF

End Module

Both RobberantLogic and antlogic were modified slightly to perform these instructions for a whole list of ants rather than a single agent.

Any methods not given pseudo code are simple(only call multiple methods or similar simple actions) and so pseudo code was not created for them with the exception of drawthingsdoublebuffering which is essentially the same as the given drawobjectDoublebuffering.

A small enhancement made was to let the user chose how many ants to create at the start of the simulation and also adding a reset button to the form so that a pre existing world can be cleared and the simulation started again.

Evaluation

The solution created fulfilled all the objectives of the assignment and the extra credit, this was done with a somewhat efficient design which meant that the program runs very well until approximately 400 ants are present in the program at which point the ants begin to move slower due to the mass amount of calculations done by each ant. This could have been improved by making some calculations not perform if the ant knew where food or a nest was (so doesn’t have to ask for it) as this would decrease the number of calculations significantly. One of the bugs in the solution is that when a food is removed from the world the ants fool each other repeatedly into thinking that there is still a food there, the ones that approach see its gone then forget but get told again where to go by another ant, this could be prevented by adding an ignore period after updating information or by having the information that food is gone sent out across the ants. This bug does sort itself out however it looks strange and is unrealistic behaviour. Another is that robber ants seem to gain speed when approaching an ant agent that is carrying food rather than approach in the correct way, this may be caused by the antAgent attempting to flee however when watching the simulation this looks as if the robber ants are attacking the antAgents so resolving the issue was never attempted. When creating a robber ant nest the amount of robber ants to be created along side it is dependant on the amount of antAgents that exist currently, due to this when the program is started with very small amounts of ants (< 10) when a robberant nest is created no robber ants will be created.

Overall I am not happy with the solution I have created, there were several efficiency issue that could have been fixed and the bugs that exist would be relatively quick fixes, these sorts of things could have been prevented with a better planning and more thoughtful process rather than the method that I took.

If doing this project again I would create at least one more class for the nests and add size properties to all existing classes. This would mean that there wouldn’t have been unnecessary global variables being used across multiple methods. There were also several inefficiency’s in the usage of the methods that were created, for example the createAnts method could have been more generalised and taken a vector as input meaning there wouldn’t have to be another method for creating robber ants and instead the random vector was created before creating the ant itself or having some sort of choice between creating at a random vector or creating at a fixed vector.