Skeleton Code Breakdown

Static Methods

Identifier / Data		Description
GetCellReference		
Parameters	**Row** : Int (passed by ref) **Column** : Int (passed by ref)	This method asks the user for the row and column values for a cell in the simulation "world". The inputs are cast to integers and assigned to the variables **Row** and **Column**. Since these variables are passed by reference rather than by value, they do not need to be returned.
Return values	n/a	
GetChoice		
Parameters	n/a	This method assigns a string input from the user to the variable **Choice** and returns that variable.
Return values	**Choice** : String	
DisplayMenu		
Parameters	n/a	This method displays the main program menu giving the 5 simulation options available to the user.
Return values	n/a	
Main		
Parameters	n/a	This method sets up the simulation. The simulation is configured using the **SimulationParameters** integer list. The method asks for user input to choose the simulation number (1–4) and uses that to select which configuration list to use when instantiating the **ThisSimulation** object. The method then enters the main program loop, calling the **DisplayMenu()** to display the main simulation options menu. The method then sets the **Choice** variable using the return from the **GetChoice()** method ranging from 1 to 4. This is used to call different methods from the **ThisSimulation** object. If the user selects 9 from the **GetChoice()** menu, the main program loop ends and the program finishes.
Return values	n/a	

Class: Simulation

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="3"><<constructor>></td></tr>
<tr><td>Parameters</td><td>SimulationParameters : Int List</td><td rowspan="2">Initialises the following protected attributes with values from the parameter SimulationParameters
• StartingNumberOfNests to the value in index 0 from the list SimulationParameters
• NumberOfRows to the value in index 1 from the list SimulationParameters
• NumberOfColumns to the value in index 2 from the list SimulationParameters
• StartingFoodInNest to the value in index 3 from the list SimulationParameters
• StartingNumberOfFoodCells to the value in index 4 from the list SimulationParameters
• StartAntsInNest to the value in index 5 from the list SimulationParameters
• NewPheromoneStrength to the value in index 6 from the list SimulationParameters
• PheromoneDecay to the value in index 7 from the list SimulationParameters

The method then initialises the two variables Row and Column and uses these to perform a nested loop to populate the list Grid with blank cells. The Grid represents the simulation “world”. Each cell is given a Row and Column value representing its position in the “world”. The bounds of this are from 1,1 to NumberOfRows, NumberOfColumns.

The method then calls the SetUpANest() method passing in the hard-coded values 2 and 4 which is the default Row,Column position of the first nest.

The method then uses a For…Next loop to add multiple nests to the simulation at random positions. The loop is governed by the variable StartingNumberOfNests as its upper bound and the variable Count, which is initialised to 2. This means that if the StartingNumberOfNests is set to 1, the loop does not run, which means that only one nest will be created.

The loop generates random values for Row and Column, using the range of 1 to NumberOfRows + 1, or 1 to NumberOfColumns + 1, to keep the values in the bounds of the simulation “world”. If the random positions selected are the same as the position of a nest already present, the program loops to select a new random position. Once a valid position has been chosen, the method calls the SetUpANest() method to create a new nest.

The method then enters another For…Next loop, this time using the lower bound of 1 and upper bound of StartingNumberOfFoodCells to add randomly positioned food cells to the simulation “world”. Food cells cannot be added into a cell containing a nest. The code uses an indefinite loop to repeatedly select random locations until a valid location is selected. The method then calls the AddFoodToCell() method to add a hard-coded amount of 500 food units to the selected cell. The method does not check if food is already present in the cell; therefore, it is possible for food to be added multiple times to the same cell.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
</table>

Identifier / Data		Description
AddFoodToNest (public)		
Parameters	**Food** : Int **Row** : Int **Column** : Int	This method uses a Foreach loop to iterate through the list of **Nests**, adding the default amount of food to a nest. If a **Nest** is at the same location as the passed-in parameters **Row** and **Column**, the method calls the **ChangeFood()** method on the **Nest** object, passing in the **Food** parameter.
Return values	n/a	
AddFoodToCell (public)		
Parameters	**Row** : Int **Column** : Int **Quantity** : Int	This method uses the **GetIndex()** method to get the index position of a cell in the **Grid** list based on its **Row** and **Column** position in the simulation "world". It then calls the **UpdateFoodInCell()** method for the cell at that index and passes it the parameter of **Quantity** to change the food level in that cell. The skeleton program often passes negative quantity values into this method to reduce the amount of food in a cell.
Return values	n/a	
AdvanceStage (public)		
Parameters	**NumberOfStages** : Int	This method contains a For…Next loop which is the main loop in the program for advancing the simulation from one stage to the next. The loop iterates from 1 to the **NumberOfStages** inclusive. Inside the loop, the method first instantiates an empty list of **Pheromones** called **PheromonesToDelete**. It then iterates through the **Pheromones** list using a Foreach loop, first calling the **AdvanceStage()** method on each pheromone and then checking each pheromone using the **GetStrength()** method to query the pheromone strength level of each. If the strength level has reached 0, the **Pheromone** is added to the **PheromonesToDelete** list. Once the loop has completed, the method then enters a second loop, iterating through the **PheromonesToDelete** list, removing all the **Pheromones** in that list from the main **Pheromones** list. This is because you can't remove an item directly from a list which you are enumerating through with a Foreach loop. Once the loop has completed, the method enters a third loop, now through the **Ants** list. It first calls the **AdvanceStage** method on each ant in the list to advance its stage. It then instantiates a local cell object called **TempCell** as a reference of the cell in the **Grid** at the location of the ant being investigated. The method then queries the ant. If it is carrying food and it is in its own nest, the method calls the **AddFoodToNest()** method, passing in the amount of food the ant is carrying together with its row and location. This ensures the food is added to the correct nest. The **UpdateFoodCarried()** method is called, passing the inverse of the amount of food the ant is carrying, which will reduce the amount of food the ant is carrying to zero.
Return values	n/a	

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="2"></td><td>If the ant, however, is in a cell which contains food, the ant currently isn't carrying any food and it has a food capacity greater than zero, the method selects a random amount of food up to the carrying capacity of the ant. AQA updated this check in version 2 of the code to prevent a new ant object (derived from the Ant class) with a food capacity of zero from attempting to pick up food. Assuming the ant can pick up food, the method uses a loop to check that the amount of food taken is not greater than what is available in the cell or greater than what the ant can carry. The method then calls the UpdateFoodInCell() on the CurrentCell to reduce the amount of food in that cell by the amount taken by the ant. Finally, it calls the UpdateFoodCarried() method on the ant to give the food to the ant to carry back to the nest.

If the ant is not in a food cell, the code then tests if it is carrying food, which means the ant is making its way back to the nest. If so, the method calls the UpdateAntsPheromoneInCell() method on the ant to reapply or generate new pheromones into the cell the ant is in. This branch of the selection statement then calls the ChooseCellToMoveTo() on the ant, passing in the results of the methods GetIndicesOfNeighbours() and GetIndexOfNeighbourWithStrongestPheromone(). This controls whether the ant is moving randomly or is following a pheromone trail.

The final task in the method is to iterate through the Nest list calling the AdvanceStage() method on each Nest in the Nests list.</td></tr>
<tr><td colspan="3">GetAreaDetails (public)</td></tr>
<tr><td>Parameters</td><td>StartRow : Int
StartColumn : Int
EndRow : Int
EndColumn : Int</td><td rowspan="3">Uses nested iteration to enumerate through part of the simulation "world" using the parameters as the top left and bottom right corners of the "window" the user wants to display from the "world".

The method uses concatenation to build up a multi-line string representing the "world". Each row in the Details string represents a single cell in the Grid. The method instantiates a local cell object called TempCell as a reference of the cell in the Grid at the location being investigated. The method then uses the GetNestInCell(), GetNumberOfAntsInCell(), GetNumberOfPheromonesInCell() and GetAmountOfFood() helper methods to build up the data for each cell.

Once the loop has completed, the Details string is returned.</td></tr>
<tr><td>Return values</td><td>Details : String</td></tr>
<tr><td colspan="2"></td></tr>
</table>

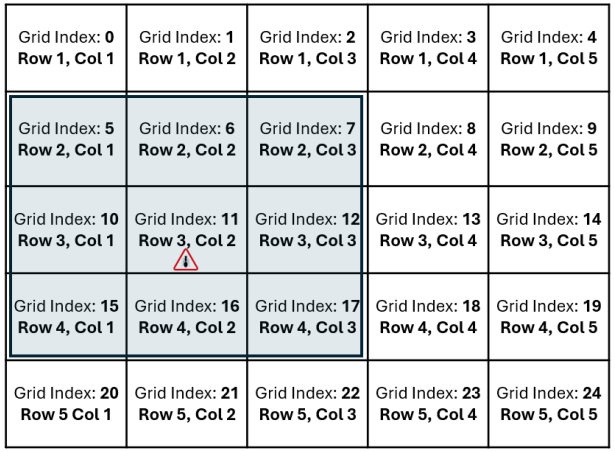
Identifier / Data		Description
GetCellDetails (public)		
Parameters	**Row** : Int **Column** : Int	This method is used to build up a more detailed string about a single cell in the simulation “world”. The method instantiates a local cell object called **CurrentCell** as a reference of the cell in the **Grid** at the location being investigated. It first calls the **GetDetails()** method on the **CurrentCell** and instantiates a copy of the **Nest** in that cell. If there is a nest present, the method calls the **GetFoodLevel()** method on that cell to find out how many food units are available in the nest. The method then uses the same technique twice. It first calls the **GetNumberOfAntsInCell()** method, passing in the **CurrentCell** object. If there are ants in the cell, the method uses iteration to find out how many ants are in that cell, adding that value to the **Details** string. It then repeats this but uses the **GetNumberOfPheromonesInCell()** method to find out which ant left those pheromones and how strong they are. Finally, the method adds some carriage returns to the string to make the output slightly easier to read and returns the string.
Return values	**Details** : String	
GetDetails (public)		
Parameters	n/a	Uses nested iteration to enumerate through the whole of the simulation “world”. The method uses concatenation to build up a multi-line string representing the “world”. Each row in the **Details** string represents a single cell in the **Grid**. The method instantiates a local cell object called **TempCell** as a reference of the cell in the **Grid** at the location being investigated. The method then uses the **GetNestInCell()**, **GetNumberOfAntsInCell()**, **GetNumberOfPheromonesInCell()** and **GetAmountOfFood()** helper methods to build up the data for each cell. Once the loop has completed, the **Details** string is returned.
Return values	**Details** : String	
GetIndex (private)		
Parameters	**Row** : Int **Column** : Int	Uses the **Row** and **Column** parameters to calculate the position of the index of a cell in the **Grid** list.
Return values	Int	

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="3">GetIndexOfNeighbourWithStrongestPheromone (private)</td></tr>
<tr><td>Parameters</td><td>Row : Int
Column : Int</td><td rowspan="3">This method first instantiates the following variables:
• StrongestPheromone to 0
• IndexOfStrongestPheromone to -1

It then uses a Foreach loop to iterate through a list of the indices of the neighbours surrounding the cell at position Row, Column. This is collected together using the GetIndicesOfNeighbours() method. If a neighbour is not valid (for example it is outside of the bounds of the simulation “world”), it is represented as -1. In the Foreach loop, the method first tests if the index of the cell is not -1 and if the strength of the pheromone in that cell is greater than the current StrongestPheromone level. If so, the method updates the IndexOfStrongestPheromone variable to the current index and also what the new StrongestPheromone variable is.

After the loop has completed, the IndexOfStrongestPheromone is returned. If no neighbours contain any pheromones, the IndexOfStrongestPheromone remains -1, which tells the ant to continue moving randomly.</td></tr>
<tr><td>Return values</td><td>IndexOfStrongestPheromone : Int</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">GetIndicesOfNeighbours (private)</td></tr>
<tr><td>Parameters</td><td>Row : Int
Column : Int</td><td rowspan="3">This method builds up and returns a list of the Grid indices of the neighbouring cells of the cell at the position Row, Column. The method uses nested iteration to look at the cells in a 3×3 grid around the cell given at position Row, Column. For example, if Row was 3 and Column was 2 in a standard 5×5 simulation “world”, the method would return a list containing [5,6,7,10,-1,12,15,16,17].

If a cell is outside the bounds of the simulation “world”, or if the cell being investigated is itself, the index is given as -1.

 
</td></tr>
<tr><td>Return values</td><td>ListOfNeighbours : Int List</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">GetNestInCell (public)</td></tr>
<tr><td>Parameters</td><td>C : Cell</td><td rowspan="3">This method iterates through the Nest list comparing the location of the Nest with the location of the parameter C. If they match, the Nest at that location is returned. If no nest is found at that location, the method gives a null return.</td></tr>
<tr><td>Return values</td><td>N : Nest</td></tr>
<tr><td colspan="2"></td></tr>
</table>

Identifier / Data		Description
GetNumberOfAntsInCell (public)		
Parameters	**C**: Cell	This method initialises an integer variable **Count** to 0. It then iterates through the **Ants** list comparing the location of the **Ant** with the location of the parameter **C**. If they match, the **Count** variable is incremented. After the loop, **Count** is returned.
Return values	**Count** : Int	
GetNumberOfPheromonesInCell (public)		
Parameters	**C**: Cell	This method initialises an integer variable **Count** to 0. It then iterates through the **Pheromones** list comparing the location of the **Pheromone** with the location of the parameter **C**. If they match, the **Count** variable is incremented. After the loop, **Count** is returned.
Return values	**Count** : Int	
GetStrongestPheromoneInCell (private)		
Parameters	**C**: Cell	This method initialises an integer variable **Strongest** to 0. It then iterates through the **Pheromones** list comparing the location of the **Pheromone** with the location of the parameter **C**. If they match, and if the **Count** variable and the strength of the **Pheromone** at that location is greater than the value in the variable **Strongest**, the value in the variable **Strongest** is replaced with the strength of the **Pheromone**. After the loop, **Strongest** is returned.
Return values	**Strongest** : Int	
SetUpANestAt (public)		
Parameters	**Row** : Int **Column** : Int	This method first adds a new **Nest** object, at the location of the parameters **Row** and **Column**, to the **Nests** list. It then adds a new **Queen**, at the location of the parameters **Row** and **Column**, to the **Ants** list. Finally, it uses a For…Next loop to add new **WorkerAnt** objects to the **Ants** list. The loop has a lower bound of 2 and the upper bound of the variable **StartingAntsInNest**. This is because the **StartingAntsInNest** must also include the **Queen**.
Return values	n/a	
UpdateAntsPheromoneInCell (public)		
Parameters	**A** : Ant	This method iterates through the **Pheromones** list. If the **Pheromone** is at the same location as the parameter **A**, and it has the same ID as parameter **A** (i.e. it belongs to the ant), then the method calls the **UpdateStrength()** method on the **Pheromone**, passing the **NewPheromoneStrength** as a parameter, and then exits the method. If the method iterates completely through the **Pheromones** list without finding a match, it means that the ant is making a new path rather than following its own original path; therefore, the method instantiates a new **Pheromone** object for the ant and adds it to the **Pheromone** list.
Return values	n/a	

Class: Cell

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int	The constructor passes the parameters **StartRow** and **StartColumn** to the base class. It then sets the protected attribute **AmountOfFood** to 0.
Return values	n/a	
GetAmountOfFood (public)		
Parameters	n/a	This is an accessor method which returns the attribute **AmountOfFood**.
Return values	**AmountOfFood** : Int	
GetDetails (public) <<override>>		
Parameters	n/a	This method is only called from the **GetCellDetails** method. It returns a concatenated string which comprises the return value from the **GetDetails()** method in the base class, followed by the **AmountOfFood** in the cell. The **GetDetails()** method in the base class, however, just returns an empty string.
Return values	String	
UpdateFoodInCell (public)		
Parameters	**Change** : Int	This is a mutator method which updates the **AmountOfFood** attribute by adding the parameter **Change** to it.
Return values	n/a	

Class: Ant

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="3"><<constructor>></td></tr>
<tr><td>Parameters</td><td>StartRow : Int
StartColumn : Int
NestInRow : Int
NestInColumn : Int</td><td rowspan="3">The constructor passes the parameters StartRow and StartColumn to the base class.

It then sets the following protected attributes:
• NestRow to the parameter NestInRow
• NestColumn to the parameter NestInColumn
• ID to the static attribute NextAntID
• Stages to 0
• AmountOfFoodCarried to 0
• FoodCapacity to 0
• TypeOfAnt to “”

Many of these attributes are updated in the classes derived from the Ant class.

The constructor also increments the static attribute NextAntID so that the next ant created has a new ID number.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">AdvanceStage (public) <<virtual>></td></tr>
<tr><td>Parameters</td><td>Nests : Nest List
Ants : Ant List
Pheromones : Pheromone List</td><td rowspan="3">This method increments the Stages attribute.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><th>Identifier / Data</th><th colspan="2">Description</th></tr>
<tr><td colspan="3">ChangeCell (protected)</td></tr>
<tr><td>Parameters</td><td>NewCellIndicator : Int
RowToChange : Int (passed by reference)
ColumnToChange : Int (passed by reference)</td><td rowspan="2">This method updates the position of the ant based on the NewCellIndicator parameter. The table on the right shows the location of neighbouring cells around an ant. The NewCellIndicator shows the index position. If the parameter is greater than 5, then the new direction must be downwards; therefore, the method increments the RowToChange parameter to move the ant downwards. If the NewCellIndicator is less than 3, the new direction must be upwards; therefore, the method decrements the RowToChange parameter to move the ant upwards.

If the NewCellIndicator is 0, 3 or 6, the new direction must be left; therefore, the method decrements the ColumnToChange parameter to move the ant left. If the NewCellIndicator is 2, 5 or 8, the new direction must be right; therefore, the method increments the ColumnToChange parameter to move the ant right. A combination of these selection statements allows the ant to move diagonally within the table shown above.

RowToChange and ColumnToChange are passed by reference; therefore, they do not need to be returned.

<table><tr><td>Index: 0</td><td>Index: 1</td><td>Index: 2</td></tr><tr><td>Index: 3</td><td>Cell containing Ant
⚠</td><td>Index: 5</td></tr><tr><td>Index: 6</td><td>Index: 7</td><td>Index: 8</td></tr></table></td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">ChooseCellToMoveTo (public) <<virtual>></td></tr>
<tr><td>Parameters</td><td>ListOfNeighbours : Int List
IndexOfNeighbourWithStrongestPheromone : int</td><td rowspan="2">This is an empty base class virtual method which is overridden in the derived classes.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">ChooseRandomNeighbour (protected)</td></tr>
<tr><td>Parameters</td><td>ListOfNeighbours : Int List</td><td rowspan="2">This method chooses the index of a neighbouring cell at random. It is called in the AdvanceStage method in the Simulation class and is used when an ant is randomly searching for food. It uses a loop to select one of the neighbours from the ListOfNeighbours parameter while the neighbour does not contain -1. If an index contains -1 it means that it is either the location of the ant, or it is a location which is outside the bounds of the simulation “world”.</td></tr>
<tr><td>Return values</td><td>Int</td></tr>
<tr><td colspan="3">GetDetails (public) <<override>></td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">This method is only called from the GetCellDetails method. It returns a concatenated string which comprises the return value from the GetDetails() method in the base class, followed by the ID of the Ant, the TypeOfAnt, and how many Stages the ant has been alive for. The GetDetails() method in the base class, however, just returns an empty string.</td></tr>
<tr><td>Return values</td><td>String</td></tr>
</table>

Identifier / Data		Description
GetFoodCapacity (public) <<override>>		
Parameters	n/a	This is an accessor method which returns the attribute **FoodCapacity**.
Return values	Int	
GetFoodCarried (public) <<virtual>>		
Parameters	n/a	This is an accessor method which returns the attribute **AmountOfFoodCarried**.
Return values	Int	
GetNestColumn (public) <<virtual>>		
Parameters	n/a	This is an accessor method which returns the attribute **NestColumn**.
Return values	Int	
GetNestRow (public) <<virtual>>		
Parameters	n/a	This is an accessor method which returns the attribute **NestRow**.
Return values	Int	
GetTypeOfAnt (public) <<virtual>>		
Parameters	n/a	This is an accessor method which returns the attribute **TypeOfAnt**.
Return values	String	
IsAtOwnNest (public) <<virtual>>		
Parameters	n/a	This returns the result of an expression which tests if the current **Row** of the ant is the same as the attribute **NestRow** and if the current **Column** of the ant is the same as the attribute **NestColumn**.
Return values	Bool	
UpdateFoodCarried (public) <<virtual>>		
Parameters	**Change** : Int	This is a mutator method which updates the **AmountOfFoodCarried** attribute by adding the parameter **Change** to it.
Return values	n/a	

Class: Pheromone

Identifier / Data		Description
<<constructor>>		
Parameters	**Row** : Int **Column** : Int **BelongsToAnt** : Int **InitialStrength** : Int **Decay** : Int	The constructor passes the parameters **Row** and **Column** to the base class. It then sets the following protected attributes: • **BelongsTo** to the parameter **BelongsToAnt** • **Strength** to the parameter **InitialStrength** • **PheromoneDecay** to the parameter **Decay**
Return values	n/a	
AdvanceStage (public) <<override>>		
Parameters	**Nests** : Nest List **Ants** : Ant List **Pheromones** : Pheromone List	This method reduces the **Strength** attribute by the value in the **PheromoneDecay** attribute. It then checks the updated **Strength** attribute. If it has gone below 0, the method resets it back to 0.
Return values	n/a	
GetBelongsTo (public)		
Parameters	n/a	This is an accessor method which returns the attribute **BelongsTo**.
Return values	Int	
GetStrength (public)		
Parameters	n/a	This is an accessor method which returns the attribute **Strength**.
Return values	Int	
UpdateStrength (public)		
Parameters	**Change** : Int	This is a mutator method which updates the **Strength** attribute by adding the parameter **Change** to it.
Return values	n/a	

Class: Nest

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="3"><<constructor>></td></tr>
<tr><td>Parameters</td><td>StartRow : Int
StartColumn : Int
StartFood : Int</td><td rowspan="3">The constructor passes the parameters StartRow and StartColumn to the base class.

It then sets the following protected attributes:
• FoodLevel to the parameter StartFood
• NumberOfQueens to 1
• ID to the static attribute NextNestID

The constructor also increments the static attribute NextNestID so that the next nest created has a new ID number.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">AdvanceStage (public) <<override>></td></tr>
<tr><td>Parameters</td><td>Nests : Nest List
Ants : Ant List
Pheromones : Pheromone List</td><td rowspan="3">This method performs a number of tasks to manage the ants and food consumption in the nest.

It first checks to see if the Ants list is null, i.e. all of the ants have died. If they have, the method exits and the nest doesn’t advance.

The method then initialises three integer variables – AntsToCull, Count and AntsInNestCount – setting them all to 0. It then iterates through the Ants list checking if the ant belongs to the nest. Note that this DOESN’T check if the ant is actually in the nest. If the ant belongs to the nest and is a Queen ant, the Count variable is incremented by 10. If the ant belongs to the nest and is a normal worker ant, the Count variable is incremented by 2 and the AntsInNestCount is incremented by 1. This is perhaps slightly misleading and the variable name WorkerAntsBelongingToNestCount would have been more descriptive. The Count variable here represents the amount of food being consumed. The preliminary material states that “The amount of food in the nest decreases at each stage by an amount determined by the number of ants that belong to the nest”, which is why the Count variable is incremented for every ant, regardless of their cell location in the simulation “world”. After looping through the Ants list, the ChangeFood() method is called, passing the inverse of the Count variable which decrements the amount of food in the nest.

The method then performs a number of checks to see if ants need to be culled. If the FoodLevel in the nest is 0 and the AntsInNestCount is greater than 0 (i.e. there are still worker ants in the simulation belonging to this nest, but not necessarily in it), the AntsToCull variable is incremented. If the FoodLevel of the nest is less than the number of worker ants belonging to this nest, the AntsToCull variable is incremented.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

Identifier / Data		Description
		The method then performs a wider check. If the **FoodLevel** in the nest is less than five times the number of worker ants belonging to this nest in the simulation, the **AntsToCull** variable is incremented. Within this scenario, the method checks if the value in the **AntsToCull** variable is greater than the number of worker ants belonging to this nest in the simulation. This is to prevent the application from attempting to remove more ants (belonging to the nest) than actually exist. The method then uses a For…Next loop, with the upper bound set to the **AntsToCull** variable, removing ants from the **Ants** list. The method uses a While loop to select ants at random from the **Ants** list. The loop will not select an ant which does not belong to the nest being inspected; however, the ant selected could be anywhere in the simulation "world", it doesn't have to actually be in the nest. This means that any ant belonging to the nest could be removed, even those located in a food cell or carrying food back to the nest. If the **Queen** is removed in this process, the **NumberOfQueens** variable is decremented. If the **FoodLevel** is not less than five times the value in the **AntsInNestCount** variable, the nest is deemed to be healthy, and therefore the **Queen** gives birth to more ants. Within this scenario, there is a 50% chance that the **Queen** will give birth to a new ant, and a 2% chance that this birth will be a new **Queen**. Combined, therefore, the chance of a new **Queen** being born is 1%. When a new **Queen** is added to the simulation, the **NumberOfQueens** attribute is incremented. New ants are added to the **Ants** list.
ChangeFood (public)		
Parameters	**Change** : Int	This method adjusts the **FoodLevel** attribute by the value in the **Change** attribute. It then checks the updated **FoodLevel** attribute. If it has gone below 0, the method resets it back to 0.
Return values	n/a	
GetFoodLevel (public)		
Parameters	n/a	This is an accessor method which returns the attribute **FoodLevel**.
Return values	Int	

Class: Entity

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int	The constructor sets the following protected attributes: • **Row** to the parameter **StartRow** • **Column** to the parameter **StartColumn** Both of these attributes are updated in the classes derived from the **Entity** class.
Return values	n/a	
AdvanceStage (public) <<virtual>>		
Parameters	**Nests** : Nest List **Ants** : Ant List **Pheromones** : Pheromone List	This is an empty base class virtual method which is overridden in the derived classes.
Return values	n/a	
GetColumn (public)		
Parameters	n/a	This is an accessor method which returns the attribute **Column**.
Return values	Int	
GetDetails (public) <<virtual>>		
Parameters	n/a	This is a base class virtual method which returns an empty string.
Return values	String	
GetID (public)		
Parameters	n/a	This is an accessor method which returns the attribute **ID**.
Return values	Int	
GetRow (public)		
Parameters	n/a	This is an accessor method which returns the attribute **Row**.
Return values	Int	
InSameLocation (public)		
Parameters	**E** : Entity	This returns the result of an expression which tests if the attribute **Row** is the same as the row of parameter **E** and the attribute **Column** is the same as the column of the parameter **E** (i.e. two entities are in the same place).
Return values	Bool	

Class: Queen

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int **NestInRow** : Int **NestInColumn** : Int	The constructor passes all four parameters to the base class. It then sets the **TypeOfAnt** attribute (from the base class) to “queen”.
Return values	n/a	

Class: WorkerAnt

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int **NestInRow** : Int **NestInColumn** : Int	The constructor passes all four parameters to the base class. It then sets the **TypeOfAnt** attribute (from the base class) to “worker” and the **FoodCapacity** attribute (from the base class) to 30.
Return values	n/a	
ChooseCellToMoveTo (public) <<override>>		
Parameters	**ListOfNeighbours** : Int List **IndexOfNeighbourWithStrongest Pheromone** : Int	This method decides how a worker ant will move based on what it is doing. It first checks if the ant is carrying any food. If it is, the method uses two IF statements to check if the **Row** and **Column** of the worker ant are less than or greater than the **NestRow** and **NestColumn** respectively. The **Row** and **Column** attributes (from the base class) are then updated accordingly to move the worker ant one cell per stage back towards the nest. Alternatively, if the worker ant is not carrying food and the value of the **IndexOfNeighbourWithStrongestPheromone** attribute is -1, this means that the worker ant is currently searching for food and there are no cells immediately neighbouring the worker ant which contain pheromone. In this case the method calls the **ChooseRandomNeighbour()** method, passing in the **ListOfNeighbours** parameter to get the index of a new cell to move to. It then calls the **ChangeCell()** method using this index to move to a new random cell. If the **IndexOfNeighbourWithStrongestPheromone** is not -1, this means that the worker ant is next to a cell or cells with pheromone in them, which will attract it. In this case the method uses the **IndexOf()** method in the built-in List class to return the index position of the **IndexOfNeighbourWithStrongestPheromone** element in the **ListOfNeighbours**. It then calls the **ChangeCell()** method using this index to move to a new strongest pheromone cell.
Return values	n/a	
GetDetails (public) <<override>>		
Parameters	n/a	This method is only called from the **GetCellDetails** method. It returns a concatenated string which comprises the return value from the **GetDetails()** method in the base class, followed by the **AmountOfFoodCarried** of the **WorkerAnt**, together with the location of its home nest.
Return values	String	

