
# Programming Tasks

These questions require you to load the **Skeleton Program** and to make programming changes to it.

*Note that any alternative or additional code changes that are deemed appropriate to make must also be evidenced, ensuring that it is clear where in the Skeleton Program those changes have been made.*

*The objective of this resource is to provide you with a selection of different questions, and solutions to those questions. Some questions are more prescriptive than others in how the task should be completed to support a range of learners. Questions which have a similar theme may use different techniques to give students a range of options on how to solve problems. Questions in this resource are not necessarily provided in an order of complexity.*

*Unless specified by the question, the solutions assume valid input and therefore don't include any error handling beyond that supplied by the original pre-release material from AQA.*

*Students are recommended to start with a clean copy of the pre-release code before attempting each of the questions in this resource. This will prevent modifications made for one question having an unintended impact on a different question.*

*These questions are based on Version 2 of the Ant Simulation code released by AQA on 21/11/25. Please ensure you are using Version 2 of the code from Centre Services.*

 

# Task 1 **Marks:** 4

This question extends the Skeleton Program to confirm that the user has selected to "Quit" the simulation. Modify the application to confirm that the user would like to end the simulation or continue when they input option 9 from the main menu. On confirmation, the program should quit and display that the simulation has ended.

**What you need to do**

## Task 1.1

Update the **Main** method. When the user selects option 9 (quit), confirm their choice. On confirmation, quit the simulation as usual and display that the simulation has ended.

## Task 1.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `9` to quit the simulation.
- Show the program correctly displaying a check to confirm that the user would like to quit.
- When prompted confirm to quit the simulation
- Show the program displaying a message confirming that the simulation has ended.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 2 — Marks: 7

This question extends the **Ant** class and the **Nest** class to show a worker ant dying of old age once it reaches the age of 14 (stages). Modify the **Ant** class to include a getter for the protected attribute **Stages** to ask an ant its age. Modify the **AdvanceStage** method in the **Nest** class to remove any ants from the **Ants** list which are older than 14 stages. This should happen at the end of the **AdvanceStage** method. Confirm to the user as each old worker ant dies.

**What you need to do**

## Task 2.1

Update the **Ant** class to include a new method which exposes the protected attribute **Stages**. Modify the **AdvanceStage** method in the **Nest** class to check if an ant is older than 14. If so, remove the ant from the **Ants** list and confirm to the user that the ant has died of old age.

## Task 2.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `1` to display five ants in the nest at the start of the simulation.
- Enter `5` followed by `15` to move the simulation on 15 stages.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing you exposing the protected attribute **Stage** and modification to the **AdvanceStage** method in the **Nest** class. [6 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 3 **Marks:** 4

This question introduces the concept of food decaying with each stage. Modify the **Simulation** class to reduce the amount of food in any cell, which contains food, by 10% with each stage to simulate food decay. The amount of food in a cell should be set so that it cannot go below zero.

### What you need to do

#### Task 3.1

Modify the **AdvanceStage** method in the **Simulation** class to reduce the amount of food in any cell by 10% with each simulation stage increment. This should happen at the start of the **AdvanceStage** method loop so that decaying food is removed before ants attempt to pick it up. Update the **UpdateFoodInCell** method as required to prevent the food level in a cell from dropping below zero.

#### Task 3.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `1` to display a total of 1500 food units shared across up to three cells.
- Enter `4` to advance the simulation by one stage.
- Enter `1` to display that the food level in each cell containing food has reduced by 10%.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **AdvanceStage** and **UpdateFoodInCell** methods in the **Simulation** class. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 4a — Marks: 4

This question extends the Skeleton Program by introducing error handling into the main menu. The supplied Skeleton Program does not give the user any feedback if they enter an invalid option choice.

**What you need to do**

### Task 4a.1

Modify the static method **GetChoice** to validate the user input for the main application menu. The method should give suitable error messages for both the input not being the correct data format or not being in range. Continue to display the menu and ask for an input until it is valid.

### Task 4a.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `"one"` to show the program giving a suitable error message that the input is not a valid integer.
- Enter `7` to show the program giving a suitable error message that the input is not in range.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **GetChoice** method. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

## Task 4b — Marks: 4

This question extends the Skeleton Program by displaying the simulation choices for the user at the start of the application and only allowing the user to select a valid simulation option. Display a menu giving details about the four simulations available. Introduce error handling so that the user can only select options from that menu.

**What you need to do**

### Task 4b.1

Write new static methods called **DisplaySimulationChoices** and **GetSimChoice**. **DisplaySimulationChoices** should display a menu detailing the simulation settings for each of the four simulations as per the pre-release document. **GetSimChoice** should validate the user input for this menu. The method should give suitable error messages for both the input not being the correct data format or not being in range. Continue to display the simulation choice menu and ask for an input until it is valid.

### Task 4b.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Show the program displaying a menu of the four simulation choices.
- Enter `"one"` to show the program giving a suitable error message that the input is not a valid integer.
- Enter `7` to show the program giving a suitable error message that the input is not in range.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **DisplaySimulationChoices** method and **GetSimChoice** method. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 4c — Marks: 2

This question extends the Skeleton Program to prevent multiple food sources from being put into the same cell when the simulation is instantiated.

**What you need to do**

**Task 4c.1**

Modify the **Simulation** constructor to introduce logic which prevents multiple food sources from being put into the same cell.

**Task 4c.2**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `1` to display three separate cells, each containing 500 units of food.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Simulation** class constructor. [1 mark]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

## Task 4d — Marks: 6

The Skeleton Program will crash if the user enters a non-numerical character when prompted to give **Row** or **Column** inputs for a cell reference. Additionally, if the user enters a numerical value which is out of range for the simulation width and height, the program will crash with an out of range error. Introduce error handling into the program to give a suitable error message to the user if they enter a value that is either in an incorrect format or out of range. The program should continue to ask the user until they enter a valid input.

**What you need to do**

**Task 4d.1**

Modify the static method **GetCellReference** to give a suitable error message if the user inputs a value that is either in an incorrect format or out of range. **Row** and **Column** should be checked individually. Modify any other required classes to operate as described.

**Task 4d.2**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `3` to select "*inspect cell*".
- When prompted, enter a row of `10`.
- Show the program giving a suitable out of range error message.
- When re-prompted, enter a row of `2`.
- When prompted, enter a column of "`ten`".
- Show the program giving a suitable format error message.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **GetCellReference** method and any helper methods to expose the **Row** and **Column** attributes. [5 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 5 **Marks:** 6

This question modifies the Skeleton Program to allow a simulation to end. In the supplied code a simulation will only end when the user selects option "9" from the main menu. Introduce new functionality to automatically end the simulation if all the ants die or all the food within the simulation runs out. Give a reason to the user.

**What you need to do**

## Task 5.1

Modify the **AdvanceStage** method in the **Simulation** class to test to see if the simulation should end and set suitable attributes to flag this and give a reason. Introduce two new methods **HasSimulationEnded** and **GetEndReason** to the **Simulation** class as getters for this data. Modify the **Main** method to test if the simulation has ended due to the reasons described. If the simulation ends, display a suitable message to the user, giving the reason, and quit the program.

## Task 5.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `15` stages. *(More stages may be required if the simulation set-up does not meet one of the requirements for ending by this stage.)*
- Show the program displaying a suitable message for why the simulation has finished.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method, **AdvanceStage** method and the **HasSimulationEnded** and **GetEndReason** getters in the **Simulation** class together with any other modifications required to operate as described. [5 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 6 **Marks:** 10

This question modifies the Skeleton Program to introduce the concept of "Flying Ant Day". Introduce a 20% chance of a queen giving birth to a flying ant when the stage is advanced. A flying ant takes three stages to mature in its own nest and has a small food capacity of five units. Once it has matured it should move to a random location within the bounds of the grid and set up a new nest. The new location should not already contain a nest. The program should tell the user which cell the flying ant has flown to. The new nest should be set up with the normal nest parameters as per the simulation. Once a flying ant has set up a new nest, the flying ant dies.

***This question only needs to operate with a simulation starting with a standard single nest.***

**What you need to do**

### Task 6.1

Create a new **FlyingAnt** class with the required methods to operate as described. Update the **Nest** class to introduce a 20% change of a **FlyingAnt** being instantiated when the stage is advanced. Modify the **AdvanceStage** method in the **Simulation** class to inform the user when a **FlyingAnt** is born and when it has flown the nest to set up a new nest. When setting up a new nest, cull the **FlyingAnt**.

### Task 6.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `5` stages. *(More stages may be required if the simulation does not instantiate a flying ant in this time.)*
- Show the program displaying a suitable message that a flying ant has been created and where it is setting up a new nest.
- Enter `4` three more times to advance the simulation three stages.
- Enter `1` to display the overall details of the simulation.
- Show the program displaying more than one nest in the simulation.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **FlyingAnt** class together with changes to any base class as required. [5 marks]
- Your PROGRAM SOURCE CODE showing the modifications to the **AdvanceStage** method in the **Simulation** class together with any other helper methods required. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 7 — Marks: 10

This question extends the Skeleton Program to introduce a single forager ant into a nest. A forager ant has a keen sense of smell and moves randomly like a normal worker ant but can smell food sources. It has a carrying capacity of 30. The smell of food should have the value of 10 times the amount of food in a cell. When all the food from a cell has been consumed, the cell should no longer smell. If a food source is one of the forager ant's neighbouring cells, the forager ant moves towards it. A forager ant collects food and takes it back to the nest as normal. The simulation should display in stages what the forager ant is doing, to make its movements easier to follow. As part of this information, when a forager ant smells food, it should tell the user what direction it is coming from using the below neighbouring cell matrix:

| | | |
|---|---|---|
| Index: **0**<br>*Direction:*<br>***North-West*** | Index: **1**<br>*Direction:*<br>***North*** | Index: **2**<br>*Direction:*<br>***North-East*** |
| Index: **3**<br>*Direction:*<br>***West*** | Cell containing Forager Ant  | Index: **5**<br>*Direction:*<br>***East*** |
| Index: **6**<br>*Direction:*<br>***South-West*** | Index: **7**<br>*Direction:*<br>***South*** | Index: **8**<br>*Direction:*<br>***South-East*** |

### What you need to do

#### Task 7.1

Create a new **ForagerAnt** class with the required methods to operate as described. Modify any base classes required to allow the **ForagerAnt** to operate. When a nest is set up, currently it adds one queen ant and an amount of worker ants. Modify this to create one less worker ant and add a **ForagerAnt**. Update the **Cell** class to introduce the concept of the smell of food. Create a new method called **GetIndexOfNeighbourWithStrongestSmell** in the **Simulation** class which returns the index of the neighbouring cell which has the strongest-smelling food in it. Use this information to direct the **ForagerAnt**.

#### Task 7.2

Display suitable messages to the user as the **ForagerAnt** moves, to inform them what it is currently doing. When the **ForagerAnt** detects a strong smell, display a message to the user to say which direction it is coming from using the matrix shown.

#### Task 7.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `5` stages. *(More stages may be required if the forager ant does not find food in this time.)*
- Enter `4` to advance the simulation one stage.
- Show the program displaying suitable messages from the forager ant as it progresses.
- Repeat the last two steps until the forager ant has shown that it has detected food, is moving towards it, and has collected food and is heading back to the nest.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **ForagerAnt** class together with changes to any base class as required. [5 marks]
- Your PROGRAM SOURCE CODE showing the changes to the **Cell** class and **AdvanceStage** method in the **Simulation** class to use the new **GetIndexOfNeighbourWithStrongestSmell** method. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 8 **Marks:** 5

This question modifies the Skeleton Program to introduce the concept of a scientist able to move individual ants from one location in the simulation to another. To comply with ethics, so not to destroy a nest, the scientist can only move a worker ant. Modify the main menu to include a new option allowing the user to relocate a single ant from one cell to another. Error handling should be included to prevent the scientist from being able to select a cell that doesn't contain any worker ants. It is assumed that the scientist will relocate the ant to a cell within the confines of the simulation.

**What you need to do**

### Task 8.1

Introduce a new menu option "6" into the main menu to allow the user to select an ant from a cell and relocate it to a new cell. Create a new **SetNewLocation** method in the **WorkerAnt** class which moves an ant to a new location. Update the base class as required.

### Task 8.2

Create a new **RelocateAnt** method in the **Simulation** class which operates as described. The modification should check that the selected cell contains a worker ant to relocate but does not need to check that the location to move to is within the range of the simulation "world". The method should give suitable feedback to the user that an ant has been successfully moved, or an error message if the user attempts to select a cell which does not contain a worker ant.

### Task 8.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `6` to enter the new menu option.
- When prompted, enter the Row `4` and the Column `1`. Enter any random, valid target location.
- Show the programing giving a suitable error message that no ant can be found at that location.
- Re-attempt the process moving from Row `2`, Column `4` to Row `2`, Column `5`.
- Show the program displaying that an ant has been successfully moved.
- Enter `1` to display the overall details of the simulation.
- Show the program displaying four ants at the nest location and one ant at Row 2, Column 5.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method together with the new **SetNewLocation** method in the **Ant** class and **RelocateAnt** method in the **Simulation** class. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 9 **Marks:** 9

This question modifies the Skeleton Program to introduce a bird predator which eats ants. A bird should be created when the simulation is instantiated. It should be placed into a random location. At each stage the bird should move to a valid neighbouring cell at random. If the bird finds any ants at that location, the bird eats a maximum of five ants; this includes at a nest, where the bird can also eat the queen. If there are fewer than five ants at a location, the bird eats all the ants in the cell. Display a suitable message of how many ants have been eaten when a bird eats ants. Modify the outputs of options 1 and 2 from the main menu to display the location of the bird in the simulation "world".

**What you need to do**

### Task 9.1

Create a new **Bird** class which inherits **Entity**. Modify the **Entity** class as required for the bird to operate as described. Put the bird into a random cell when the simulation starts. Modify the **AdvanceStage** method in the **Simulation** class to move the bird around the simulation "world". At each stage, if the bird moves into a cell containing ants, remove five ants at that location. If there are fewer than five ants at that location, remove them all.

### Task 9.2

Modify the **GetDetails** and **GetAreaDetails** methods to show when the bird is present in a cell. Create any helper methods required for this to operate. When a bird eats ants, display a message showing what cell it is in and how many ants it has eaten.

### Task 9.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `5` stages. *(More stages may be required if the bird does not find any ants in this time.)*
- Repeatedly enter `4` to advance the simulation one stage until the bird finds a cell with ants.
- Show the program displaying a suitable message of how many ants the bird has eaten.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **Bird** class together with changes to any base class and **Simulation** class to enable it to operate as required. [5 marks]
- Your PROGRAM SOURCE CODE showing the changes to the **AdvanceStage** method in the **Simulation** class. [1 mark]
- Your PROGRAM SOURCE CODE showing the changes to the **GetDetails** and **GetAreaDetails** methods in the **Simulation** class together with any helper methods. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 10 — Marks: 9

This question modifies the Skeleton Program to introduce a storm weather event which destroys all ants and pheromones in any cell it occurs in (including the queen if the weather event occurs on a nest). Only one weather event can occur in the simulation at a time. A weather event lasts for three stages before it expires. The weather event has a 75% chance of occurring in any stage. It can occur in any cell in the simulation, including those containing a nest (but it does not destroy the nest). Worker ants do not like bad weather; therefore, when they are searching randomly for food, they will avoid a cell containing the storm. If, however, they are carrying food, their instinct to return it to the nest is so great that they will return to the nest regardless of the location of the weather. If an active weather event is happening at a nest location and an ant returns to the nest, it is killed. The location of the weather event should be displayed on the details and area details of the simulation "world" to allow its position tracking.

**What you need to do**

### Task 10.1

Create a new **Weather** class which inherits the **Entity** class. Update the **AdvanceStage** method in the **Simulation** class to have a 75% chance of a weather event occurring in a stage. Include the **AdvanceStage** method in the **Weather** class to remove all ants and pheromones at the same location as the storm. While the storm is active, ants which are moving randomly should avoid the cell containing the storm, unless they are carrying food back to the nest, in which case they should also be removed if they enter the cell containing the storm.

### Task 10.2

Modify the **GetDetails** and **GetAreaDetails** methods to show when the weather event is present in a cell. When ants or pheromones are destroyed by a storm, display a message showing how many of each have been destroyed.

### Task 10.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Repeatedly enter `4` to advance the simulation one stage until a weather event occurs.
- If the weather event has occurred in a cell containing ants or pheromones, show the program displaying a suitable message of how many ants/pheromones have been removed. *(Repeat the last two steps if the weather event occurs in an empty cell.)*

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **Weather** class together with changes to any base class and **Simulation** class to enable it to operate as required. [5 marks]
- Your PROGRAM SOURCE CODE showing the changes to the **AdvanceStage** method in the **Simulation** class. [2 marks]
- Your PROGRAM SOURCE CODE showing the changes to the **GetDetails** and **GetAreaDetails** methods in the **Simulation** class. [1 mark]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 11 **Marks:** 5

This question introduces the concept of tracking data into the Skeleton Program to allow scientists to monitor the efficiency of individual ants. The scientists are interested in how many food units each individual ant delivers together with the average amount delivered by all the worker ants within a nest. The statistics should continue to show data about an ant even after it has been culled.

**What you need to do**

### Task 11.1

Create a new option "6" into the main menu which displays the food delivery stats. Use a suitable data structure in the **Simulation** class to store how many food units each worker ant is individually delivering back to the nest. Modify the **AdvanceStage** method in the **Simulation** class to update the data structure. Create a new **DisplayFoodDeliveryData** method in the Simulation class which operates as described.

### Task 11.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `10` stages.
- Enter `6` to display the delivery data.
- Show the program displaying correct data for each ant in the simulation together with the average delivery per ant for all the ants in the colony.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method, the **AdvanceStage** method in the **Simulation** class and the new **DisplayFoodDeliveryData** method. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 12 — Marks: 7

This question modifies the Skeleton Program by allowing the user to enter their own simulation parameters rather than using one of the four built-in simulations. Introduce a new menu at the start of the application which allows the user to input the parameters for a custom simulation. The new menu should ask the user to enter a value of 1–4 to select an inbuilt simulation, or 5 to select a custom simulation. The modifications should inform the user what the bounds are for their input. To ensure that the simulation runs reliably, only accept integer inputs and use the following range checks:

| Simulation Parameter | Min Value (inclusive) | Max Value (inclusive) |
|---|---|---|
| Number of nests | 1 | 5 |
| Number of rows | 5 | 20 |
| Number of columns | 5 | 20 |
| Starting number of food units in each nest | 100 | 1000 |
| Starting number of food cells | 1 | 10 |
| Starting number of ants in each nest | 5 | 20 |
| Pheromone strength | 500 | 2000 |
| Pheromone decay level per stage | 10 | 200 |

**What you need to do**

**Task 12.1**

Create a new static method **GetCustomSimulation** which prompts the user to enter in the required simulation parameters as described. Include any required helper methods to ensure that the data supplied to the simulation is in the correct format and is within range. Give appropriate error messages to the user and allow them to re-enter their data until it is valid.

**Task 12.2**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `5` to select a custom simulation.
- When prompted to choose how many nests to include in the simulation, enter `-1`.
- Show the program displaying a suitable error message that the value is not in range.
- When prompted, enter in the following custom simulation data:

| Simulation Parameter | Value to Test |
|---|---|
| Number of nests | 2 |
| Number of rows | 5 |
| Number of columns | 6 |
| Starting number of food units in each nest | 750 |
| Starting number of food cells | 4 |
| Starting number of ants in each nest | 18 |
| Pheromone strength | 500 |
| Pheromone decay level per stage | 100 |

- Enter `1` to display the overall simulation details.
- Show a second screenshot of the program displaying the custom simulation details. *(As per the original code, it is possible that some of the food sources could be in the same cell as each other.)*

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method. [1 mark]
- Your PROGRAM SOURCE CODE showing the new **GetCustomSimulation** method together with any required helper methods. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. *(Two screenshots required.)* [2 marks]

 

# Task 13 **Marks:** 4

This question adapts the Skeleton Program by introducing the concept of a custom starting amount of food in food cells at the start of the simulation. Add a prompt at the start of the application to ask the user how large the food units should be. Pass this in as an additional simulation parameter to allow the user to customise the simulation.

**What you need to do**

### Task 13.1

Add a new prompt at the start of the application and modify the **Simulation** class constructor to operate as described.

### Task 13.2

Test that the changes you have made work:

- Run the Skeleton Program.
- When prompted to enter the starting amount of food in food cells, enter `800`.
- Enter `1` to start simulation 1.
- Enter `1` to display the overall simulation details.
- Show the program displaying the custom simulation details. *(As per the original code, it is possible that some of the food sources could be in the same cell as each other.)*

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Simulation** class together with the changes to the **Main** method. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 14 **Marks:** 13

This question adapts the Skeleton Program by introducing a soldier ant. A nest should contain the same number of soldier ants as normal worker ants. A soldier ant has a small food capacity of five units. Introduce a new menu option to allow the user to place a predator into a cell in the simulation. The simulation can have multiple predators. Soldier ants move out of the nest just like normal worker ants but then stay still in a state of "*patrolling*". If a predator is in one of the neighbouring cells of a soldier ant, the soldier ant should move into the same cell as the predator and change its state to "*engaging*". When "*engaging*", the soldier ant should release pheromones which are five times the strength of a normal pheromone. Other soldier ants, and worker ants not carrying food back to the nest, should be attracted to this pheromone as the strongest around them. The eventual outcome would be that the predator is overwhelmed; however, this does not need to be simulated.

**What you need to do**

### Task 14.1

Create a new **Predator** class which inherits **Entity**. Add a new option "6" to the main menu to allow the user to place a **Predator** at any location in the simulation "world". Confirm to the user where the **Predator** has been placed.

### Task 14.2

Create a new **SoldierAnt** class which inherits **Ant**. Include a suitable attribute to store the current state of the **SoldierAnt**. Modify any required methods in the base class to allow the **SoldierAnt** to operate as described. Modify the **AdvanceStage** and **UpdateAntsPheromoneInCell** methods in the **Simulation** class to operate as described. Include suitable feedback messages to the user to inform them when a **SoldierAnt** moves from "*patrolling*" to "*engaging*" and when it is in combat with the predator.

### Task 14.3

Update the **GetDetails** method in the **Simulation** class to separately show how many ants (worker and queen) and soldier ants are in a cell AND how many predators are in the cell. Create any helper methods required to do this.

### Task 14.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `4` to advance the simulation by one stage.
- Show the program displaying simulation details showing soldier ants are patrolling in the cells surrounding the nest. (Any combination of cells: 1:3, 1:4, 1:5, 2:3, 2:5, 3:3, 3:4, 3:5.)
- Enter 6 to place a predator into the simulation.
- Select a cell location next to a soldier ant.
- Enter `4` to advance the simulation by one stage.
- Enter `1` to display the overall simulation details.
- Show the program displaying that a soldier ant is / soldier ants are engaging with the predator *(first screenshot)*.
- Enter `5` to advance multiple stages.
- When prompted, enter `5` stages.
- Enter `1` to display the overall simulation details.
- Show the program displaying that a soldier ant is / soldier ants are fighting with the predator *(second screenshot)*.

 

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the new **SoldierAnt** class together with changes to any base class to enable it to operate as required. [2 marks]
- Your PROGRAM SOURCE CODE showing the new **Predator** class together with changes to any base class or **Simulation** class to enable it to operate as required. [2 marks]
- Your PROGRAM SOURCE CODE showing the changes to or creation of new methods in the **Simulation** class to enable the solution to operate as required. [5 marks]
- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method. [1 mark]
- Your PROGRAM SOURCE CODE showing the changes to the **GetDetails** method in the **Simulation** class together with any helper methods required to enable it to operate as described. [1 mark]
- SCREEN CAPTURE(S) showing the required tests. [2 marks]

 

## Task 15 **Marks:** 6

This question extends the Skeleton Program by providing a summary of the simulation's status. Create new functionality which provides the following output in a simple, easy to read format on screen:

- Total number of ants currently in the simulation.
- Number of ants currently carrying food in the simulation.
- Total amount of food currently stored at all nests in the simulation.
- Average ant age (stages alive) of all the ants currently in the simulation.

**What you need to do**

**Task 15.1**

Add a new option "6" to the main menu to allow the user to display the colony summary stats. Create a new method **GetColonySummary** in the **Simulation** class which operates as described. Create any required helper methods in the various classes to support this.

**Task 15.2**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `4` to start simulation 4.
- Enter `5` to advance multiple stages.
- When prompted, enter `10` stages.
- Enter `6` to select the new **GetColonySummary** method.
- Show the program displaying the colony summary data.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method together with the new **GetColonySummary** method and any helper methods required. [5 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 16 **Marks:** 5

This question extends the Skeleton Program by providing a heat map of the current levels of pheromone in the cells within the simulation "world". Create new functionality which draws the simulation "world" as a 2D grid/table showing the total pheromone strength in each cell. The new functionality should work for any sized simulation "world".

**What you need to do**

**Task 16.1**

Add a new option "6" to the main menu to allow the user to display the simulation "world" as a 2D grid/table. Create a new method **DisplayPheromoneHeatMap** in the **Simulation** class which displays the total pheromone strength of each cell in the grid. Create a method **GetTotalPheromoneStrengthInCell** in the **Simulation** class that takes in a cell as a parameter and returns the total strength of pheromone in that cell.

**Task 16.2**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `30` stages.
- Enter `6` to select the new **DisplayPheromoneHeatMap** method.
- Show the program displaying the heat map. *(The simulation may need to be advanced by more stages to provide sufficient data to display.)*

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method together with the new **DisplayPheromoneHeatMap** and **GetTotalPheromoneStrengthInCell** methods. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 17 — Marks: 7

This question extends the Skeleton Program to include the concept of disease affecting ants. Each ant should be given the status of "diseased" or "healthy". New functionality should be added to allow the user to select an individual cell to infect. All worker ants in the selected cell are infected. Queens cannot be infected. Once a cell has been selected, the program should inform the user how many ants have been infected in the selected cell. Diseased ants will die 10 stages after being infected. The program should inform the user when this happens. Once an ant becomes infected, it cannot move.

**What you need to do**

### Task 17.1

Add a new option "6" to the main menu to allow the user to infect ants in a particular cell. Create the new method **InfectAntsInCell** in the **Simulation** class which operates as described. Modify the **Ant** class to allow the functionality described. Modify the **AdvanceStage** method in the **Nest** class to remove any ants which have been diseased for at least 10 stages. Inform the user how many ants have died.

### Task 17.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `10` stages.
- Enter `6` to select the new **InfectAntsInCell** method.
- When prompted, select a cell which contains multiple ants.
- Enter `5` to advance multiple stages.
- When prompted, enter `12` stages.
- Enter `1` to display the overall simulation details.
- Show the program displaying how many ants were originally infected and how many have been removed.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method. [1 mark]
- Your PROGRAM SOURCE CODE showing the new **InfectAntsInCell** method in the **Simulation** class together with modifications to the **Ant** class and **Nest** class required to enable it to operate as described. [5 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 18 — Marks: 8

This question extends the Skeleton Program to introduce a **DangerPheromone** which is released within a nest when the food level drops too low. The **DangerPheromone** has a strength of 100 and a decay rate of 10. The **DangerPheromone** does not belong to a particular ant (it has the ID of 0) but is instead a composition relationship with the **Nest** class. When the food level in a nest drops below 50 units, the **DangerPheromone** is released. Introduce a new menu option which allows the user to view the nests with dangerously low food levels. Additionally, introduce new functionality to highlight dangerously low food levels when viewing the simulation in normal detail view, area detail view, and individual cell detail view.

**What you need to do**

### Task 18.1

Create a new **DangerPheromone** class which inherits **Pheromone**. Modify the **Nest** class to include a **DangerPheromone** attribute. Modify the **AdvanceStage** in the **Simulation** class to check the food levels in each nest. If the food level drops below 50 units, mark the nest as in danger. If the food level in the nest rises back up again, mark the nest as no longer in danger.

### Task 18.2

Add a new option "6" to the main menu to call a new **GetDangerPheromoneLocations** method in the **Simulation** class. The new method should list the locations of nests with low food levels together with how much food is left in them.

Additionally, modify the **GetDetails**, **GetAreaDetails** and **GetCellDetails** methods to highlight when a nest is dangerously low on food. Create any required helper methods in the various classes to support this.

### Task 18.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `4` to start simulation 4.
- Enter `5` to advance multiple stages.
- When prompted, enter `30` stages.
- Enter `6` to select the new **GetDangerPheromoneLocations** method. *(The simulation may need to be advanced by more stages to provide sufficient data to display.)*
- Enter `3` to inspect an individual cell.
- When prompted, enter the row and column of a nest shown to be in danger.
- Show the program displaying the location and food levels of nests which are dangerously low on food and then displaying the details of one of the nests in danger.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method. [1 mark]
- Your PROGRAM SOURCE CODE showing the new **DangerPheromone** class together with changes to any base class or connected classes to enable it to operate as required. [2 marks]
- Your PROGRAM SOURCE CODE showing the changes to the **AdvanceStage** method in the **Simulation** class. [1 mark]
- Your PROGRAM SOURCE CODE showing the new **GetDangerPheromoneLocations** method in the Simulation class. [1 mark]
- Your PROGRAM SOURCE CODE showing the changes to the **GetDetails**, **GetAreaDetails** and GetCellDetails methods in the **Simulation** class together with any helper methods. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

# Task 19 **Marks:** 6

Ants dislike cinnamon because of its strong scent, particularly the compound cinnamaldehyde, which disrupts their ability to follow pheromone trails and navigate. This question extends the Skeleton Program to introduce cinnamon into a cell. The cinnamon overrides the pheromone, removing it from the cell. An ant looking for food will not enter a cell containing cinnamon. The scent of cinnamon is so strong it also removes all of the food from a cell, making the cell unappealing to a worker ant looking for food. Although an ant already in a cell containing cinnamon doesn't die, it cannot update the pheromones in the cell or add new pheromones. Introduce a new menu option to allow the user to drop cinnamon into a cell.

**What you need to do**

**Task 19.1**

Add a new option "6" to the main menu to call the new **RemovePheromonesFromCell** method in the **Simulation** class which places cinnamon into a cell and removes all of the pheromones from it. *(No error handling is required to check the validity of the cell selected by the user.)*

Modify the **Cell** class to identify when cinnamon has been placed in it.

**Task 19.2**

Adjust the required methods to prevent a worker ant from entering or adding pheromones to a cell containing cinnamon.

**Task 19.3**

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `1` to start simulation 1.
- Enter `5` to advance multiple stages.
- When prompted, enter `15` stages.
- Enter `6` to select the new **RemovePheromonesFromCell** method.
- When prompted, enter the row and column of a cell containing high levels of pheromone.
- Enter `1` to display the overall simulation details.
- Show the program displaying all the pheromones removed from the selected cell.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **Main** method and modification to the Cell class. [2 marks]
- Your PROGRAM SOURCE CODE showing the new **RemovePheromonesFromCell** method in the **Simulation** class. [1 mark]
- Your PROGRAM SOURCE CODE showing the modifications to any required methods to prevent a worker ant from entering a cell containing cinnamon or updating pheromones / adding new pheromones to the cell. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 

## Task 20 — Marks: 5

In the Skeleton Program, worker ants move randomly looking for food. This could continue indefinitely. This question modifies the program to introduce the idea of backtracking once after six steps of unsuccessful searching. If a worker ant does not find a food source or pheromone trail within six steps it should backtrack through its route all the way back to the nest, before resetting and going back out again to forage for food. When backtracking, the ant should inform the user of its ID and that it is backtracking and which cell it is moving to in each stage so the user can monitor its progress. If the worker ant finds food or a pheromone trail within the six stages, the movement history should be reset.

**What you need to do**

### Task 20.1

Add a suitable data structure to the **WorkerAnt** class to store the position history of the worker ant as it moves through the simulation "world". Modify the **ChooseCellToMoveTo** method in the **WorkerAnt** class to add to the data structure as the worker ant moves from cell to cell, operating as described.

Create a new method **BacktrackToPreviousCell** in the **WorkerAnt** class which moves the worker ant back to its cell location from the previous stage. Stop when the worker ant reaches its nest.

### Task 20.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `4` to start simulation 4.
- Enter `5` to advance multiple stages.
- When prompted, enter `7` stages.
- Show the program displaying backtracking route(s) of ants which have not successfully found food. *(The simulation may need to be advanced by more stages to provide sufficient data to display.)*

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE showing the modifications to the **WorkerAnt** class. [2 marks]
- Your PROGRAM SOURCE CODE showing the addition of a new **BacktrackToPreviousCell** method to the **WorkerAnt** class. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

 