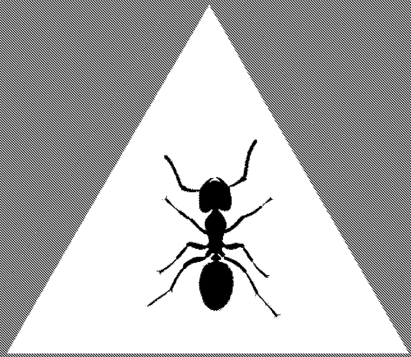
Computer Science

A Level | AQA | 7517

2015 specification
for the 2026 exam



Ant Simulation

PAPER 1 EXAM RESOURCE PACK 2026

for A Level AQA Computer Science

JAVA EDITION

- DIGITAL RESOURCE -

This pack includes paper versions of the electronic files.

Go to **zzed.uk/ProductSupport** to download the electronic files.

zigzageducation.co.uk

POD 12894

Contents

** The electronic PDF versions of these files are password-protected, so that students can only access them with your permission. Passwords can be found in the Teacher's Introduction on page iv.*

*** Note there are also electronic copies of the UML Diagrams ('Complete' & 'Activity' versions) provided.*

Teacher's Introduction

The resource pack consists of the following sections:

- **Code breakdown**: a detailed technical overview of the Skeleton Program, describing in detail each class and method in turn – including their purpose/function, parameters and return values. Note that this is intended as a helpful reference document only, and not as a substitute for exploring the code in a practical manner. There is also a 'Grid Example' file which can be used as a printout to help students to understand the simulation.
- **UML class diagram activity**: requires you to study the program and fill in the gaps with the missing class/method names, data types, associations and access levels.
- **Video**: a quick overview of the ***Ant Simulation*** application mechanics – intended as a visual aid to accompany the notes in the official AQA pre-release material.
- **Theory questions**: designed to test your understanding of the Skeleton Program. These questions require access to the program, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on versions are available.
- **Coding tasks**: there are 24 modification tasks to test your programming skills – as well as an additional 15 modification ideas that you may also want to try as extension tasks.
- **Solutions / Mark schemes**: included for UML class diagram activity, theory questions, and coding tasks.

This resource is intended to supplement your teaching only. **Please read full disclaimer (p. iii) before using it.**

DIGITAL RESOURCE

Once you have downloaded the files for this resource via (**zzed.uk/ProductSupport**) you will have access to the following:

`AntSimulation` this folder contains all of the content (PDF/DOCX) accessible via a HTML interface
`Passwords.txt` for teacher use – this file contains all of the passwords for the protected PDFs (also listed below)

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

Installation: Extract the files from the downloaded ZIP file and move the entire `TargetClear` folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

Passwords: All of the PDFs accessible via the *Solutions* web page are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

This pack is 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.

Ant Simulation

Ant Simulation represents a small world of ants showing their interaction with nests, food and pheromone trails.

The user can experiment with four different simulations, each representing different configurations of the small world. The world is represented by either a 5×5 or 10×10 grid. Simulations 1–3 contain a single ant nest, and simulation 4 contains two ant nests. At the start of the simulation, each nest contains a single queen ant and multiple worker ants (configured by parameters when the application starts up).

The application uses the concept of stages to advance, each stage essentially being a snapshot of the current state of the simulation world. As the user advances the stages, the entities within the simulation world, be those ants, pheromones or the nests themselves, all advance one stage at a time, each performing one of their preset operations. Examples of these operations are move, pick up food, follow a pheromone trail, or deposit food at the nest.

Initially, worker ants move at random around the simulation world, moving to one of their neighbouring cells per stage. If an ant finds a cell containing food, the ant picks up a small amount of that food, and then as the stages are advanced by the user, makes its way directly back to the nest. As it moves back towards the nest, the worker ant lays down a pheromone trail. If other worker ants are near to that pheromone trail, they will follow it to the food source to also collect food and take it back to the nest. This represents how ants behave in the real world.

The application also represents the relationship between food available at a nest and ant population. At each stage, if the food in a nest gets too low, or the population gets too high for the food available, some of the ants belonging to the nest are culled.

When the nest has plentiful food levels, however, the queen ant will give birth to new worker ants and even on a very small chance, another queen.

During operation, the user can display the simulation world in its entirety, view part of it, or even view an individual cell within the world to see it in detail.

The objective of the application is to experiment with different configurations of the simulation world, advancing the simulation stage by stage to see how the ants interact with each other, the food around them, and the nest.

This resource aims to help you get to grips with and prepare for the A Level Paper 1 examination for summer 2026, which is partly based on the ***Ant Simulation*** pre-release material.

Ant Simulat

Skeleton Code Breakdo

Static Class: IntWrapper

Identifier / Data		Description
<<constructor>>		
Parameters	initialValue : Int	This static class is used to w public scope within the applic the application to move the R This technique is needed bec Therefore, using this techniq variable public or passing it b

Static Methods

Identifier / Data		Description
getCellReference		
Parameters	**row** : Int (passed by IntWrapper Object reference) **column** : Int (passed by IntWrapper Object reference)	This method asks the user for the row a inputs are cast to integers and assigned passed as static IntWrapper object ra
Return values	n/a	
getChoice		
Parameters		This method assigns a string input from
Return values	**choice** : String	
displayMenu		
Parameters	n/a	This method displays the main program

Identifier / Data		Description
Return values	n/a	
main		
Parameters	n/a	This method sets up the simulation. The integer... The method asks for user in select which configuration list to use wh The method then enters the main progr simulation options menu. The method t **getChoice()** method ranging from 1 to 4 **thisSimulation** object. If the user selects 9 from the **getChoice**
Return values	n/a	

Class: *Simulation*

Identifier / Data		Description
<<constructor>>		
Parameters	**simulationParameters** : Int List	Initialises the following protected att **simulationParameters** • **startingNumberOfNests** to the • **numberOfRows** to the value i • **numberOfColumns** to the valu • **startingFoodInNest** to the valu • **startingNumberOfFoodCells** • **startAntsInNest** to the value i • **newPheromoneStrength** to th • **pheromoneDecay** to the value The method then initialises the two nested loop to populate the list grid Each cell is given a **row** and **colum** this are from 1,1 to **numberOfRows** The method then calls the **setUpAN** is the default **row,column** position
Return values	n/a	

Identifier / Data		Description
		The method then uses a For…Next positions. The loop is governed by t the variable count, which is initialis to 1, the loop does not run, which m The loop generates random values + 1, or 1 to numberOfColumns + 1 the random positions selected are t program loops to select a new rand method calls the setUpANest() me The method then enters another Fo bound of startingNumberOfFood "world". Food cells cannot be addec to repeatedly select random locatio addFoodToCell() method to add a method does not check if food is al added multiple times to the same c
addFoodToNest (public)		
Parameters	food : Int row : Int column : Int	This method uses a Foreach loop t food to a nest. If a nest is at the sa the method calls the changeFood()
Return values	n/a	
addFoodToCell (public)		
Parameters	row : Int column : Int quantity : Int	This method uses the getIndex() me its row and column position in the s for the cell at that index and passes cell. The skeleton program often pas amount of food in a cell.
Return values	n/a	

Identifier / Data		Description
advanceStage (public)		
Parameters	**numberOfStages** : Int	This method contains a For…Next lo simulation f[illegible] one stage to the nex Inside the loop, the method first insta pheromonesToDelete. It then iterat calling the **advanceStage()** method the **getStrength()** method to query t reached 0, the **pheromone** is added Once the loop has completed, the m **pheromonesToDelete** list, removin list. This is because you can't remov through with a Foreach loop. Once the loop has completed, the m the **advanceStage** method on each cell object called **tempCell** as a refe investigated. The method then queri method calls the **addFoodToNest(** together with its row and location. Th **updateFoodCarried()** method is ca carrying, which will reduce the amou If the ant, however, is in a cell which a food capacity greater than zero, th capacity of the ant. AQA updated thi *(derived from the Ant class)* with a fo the a[illegible] pi[illegible] up food, the method greater than what is available in the c calls the **updateFoodInCell()** on the amount taken by the ant. Finally, it ca food to the ant to carry back to the ne If the ant is not in a food cell, the cod its way back to the nest. If so, the me ant to reapply or generate new phero statement then calls the **chooseCell** **getIndicesOfNeighbours()** and ge controls whether the ant is moving ra The final task in the method is to iter each **nest** in the **nests** list.
Return values	n/a	

Identifier / Data		Description
getAreaDetails (public)		
Parameters	**startRow** : Int **startColumn** : Int **endRow** : Int **endColumn** : Int	Uses nested iteration to enumerate the top [illegible] bottom right corners The method uses concatenation to in the **details** string represents a si called **tempCell** as a reference of th then uses the **getNestInCell()**, **get** and **getAmountOfFood()** helper me Once the loop has completed, the d
Return values	**details** : String	
getCellDetails ([illegible])		
Parameters	**row** : Int **column** : Int	This method is used to build up a m The method instantiates a local cell the location being investigated. It firs instantiates a copy of the **nest** in tha **getFoodLevel()** method on that cel The method then uses the same tec method, passing in the **currentCell** find out how many ants are in that c uses the **getNumberOfPheromone** and how strong they are. Finally, the method adds some carri [illegible] returns the string.
Return values	**details** : String	
getDetails (public)		
Parameters	n/a	Uses nested iteration to enumerate The method uses concatenation to in the **details** string represents a si called **tempCell** as a reference of th then uses the **getNestInCell()**, **get** and **getAmountOfFood()** helper me Once the loop has completed, the d
Return values	[illegible]	

Identifier / Data		Description
getIndex (private)		
Parameters	**row** : Int **column** : Int	Uses the **row** and **column** paramet
Return values	Int	
getIndexOfNeighbourWithStrongestPh[illegible] (private)		
Parameters	**row** : [illegible] [illegible] : In[illegible]	This method first instantiates the follo • **strongestPheromone** to 0 • **indexOfStrongestPheromone**
Return value	[illegible]**dexOfStrongestPheromone** : Int	It then uses a Foreach loop to iterate cell at position **row**, **column**. This is method. If a neighbour is not valid (f it is represented as -1. In the Foreach if the strength of the pheromone in th If so, the method updates the **index** what the new **strongestPheromone** After the loop has completed, the **in** contain any pheromones, the **index** continue moving randomly.
getIndicesOfNeighbours (private)		
Parameters	**row** : Int **column** : Int	This meth[illegible]ilds up and returns a [illegible]s o[illegible]he neighbouring cells at t[illegible]sition **row**, **column**. The me [illegible]sted iteration to look at the cells in around the cell given at position **row** For example, if **row** was 3 and **colu** in a standard 5×5 simulation "world", method would return a list containing [5,6,7,10,-1,12,15,16,17]. If a cell is outside the bounds of the s "world", or if the cell being investigate the index is given as -1.
Return values	**listOfNeighbours** : Int List	

Identifier / Data		Description
getNestInCell (public)		
Parameters	c : Cell	This method iterates through the ne[illegible] parameter [illegible] they match, the nest [illegible] the [illegible] gi[illegible]s a null return.
Return values	n : Nest	
getNumberOfAntsInCell (public)		
Parameters	c: Cell	This method initialises an integer va[illegible] the location of the ant with the locati[illegible] incremented. After the loop, count i[illegible]
Return values	co[illegible]	
getNumber[illegible]on[illegible]esInCell (public)		
Parameters	c: Cell	This method initialises an integer var[illegible] comparing the location of the phero[illegible] count variable is incremented. After
Return values	count : Int	
getStrongestPheromoneInCell (private)		
Parameters	c: Cell	This method initialises an integer var[illegible] list comparing the location of the ph[illegible] if the count variable and the strengt[illegible] the variable strongest, the value in [illegible] pheromone. After the loop, stronge[illegible]
Return values	strongest : Int	
setUpANestAt (public)		
Parameters	row : Int column : Int	This met[illegible]st adds a new Nest o[illegible] nes[illegible] th[illegible]n adds a new Queen ant[illegible] [illegible]nally, it uses a For…Next [illegible] a lower bound of 2 and the uppe[illegible] the startingAntsInNest must also i[illegible]
Return values	n/a	
UpdateAntsPheromoneInCell ([illegible])		
Parameters	[illegible]	This method iterates through the ph[illegible] parameter a, and it has the same ID [illegible] the updateStrength() method on th[illegible] parameter, and then exits the metho[illegible] list without finding a match, it means [illegible] original path; therefore, the method i[illegible] the pheromone list.
Return value[illegible]	n/a	

Class: Cell

Identifier / Data		Description
<<constructor>>		
Parameters	**startRow** : Int **startColumn** : Int	The constructor passes the parame It then sets the protected attribute a
Return values	n/a	
getAmountOfFood (public)		
Parameters		This is an accessor method which r
Return values	**amountOfFood** : Int	
getDetails (public) <<override>>		
Parameters	n/a	This method is only called from the g comprises the return value from the g **amountOfFood** in the cell. The getD empty string.
Return values	String	
updateFoodInCell (public)		
Parameters	**change** : Int	This is a mutator method which upda **change** to it.
Return values	n/a	

Class: Ant

Identifier / Data		Description
<<constructor>>		
Parameters	startRow : Int startColumn : Int nestInRow : Int nestInColumn : Int	The constructor passes the parame It then sets the following protected a • nestRow to the parameter nes • nestColumn to the parameter • id to the static attribute nextA • stages to 0 • amountOfFoodCarried to 0 • foodCapacity to 0 • typeOfAnt to "" Many of these attributes are update The constructor also increments the new ID number.
Return values	n/a	
advanceStage (public) <<virtual>>		
Parameters	nests : Nest List ants : Ant List pheromones : Pheromone List	This method increments the stages
Return values	n/a	

Identifier / Data		Description
changeCell (protected)		
Parameters	**newCellIndicator** : Int **rowToChange** : Int (passed by IntWrapper Object reference) **columnToChange** : Int (passed by IntWrapper Object reference)	This method updates the position of t on the **newCellIndicator** parameter. the [illegible] the location of neighb around the ant. The **newCellIndicat** index position. If the parameter is gre then the new direction must be down therefore, the method increments the **rowToChange** parameter to move t downwards. If the **newCellIndicator** the new direction must be upwards; t method decrements the **rowToChan** If the **newCellIndicator** is 0, 3 or 6, t the **columnToChange** parameter to direction must be right; therefore, the the ant right. A combination of these s table shown above. **rowToChange** and **columnToChar** do not need to be returned.
Return values	n/a	
chooseCellToMoveTo (public) <<virtual>>		
Parameters	**listOfNeighbours** : Int List **indexOfNeighbourWithStrongestPheromone** : int	This is an empty base class virtual m
Return values	n/a	
chooseRandomNeighbour (protected)		
Parameters	**listOfNeighbours** : Int List	This method chooses the index of a method in the **simulation** class and loop to select one of the neighbours does not contain -1. If an index con a location which is outside the boun
Return values	Int	
getDetails (public) <<override>>		
Parameters	n/a	This method is only called from the comprises the return value from the the **ant**, the **typeOfAnt**, and how m method in the base class, however,
Return values	String	

Identifier / Data		Description
getFoodCapacity (public) <<override>>		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getFoodCarried (public) <<virtual>>		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getNestColumn (public) <<virtual>>		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getNestRow (public) <<virtual>>		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getTypeOfAnt (public) <<virtual>>		
Parameters	n/a	This is an accessor method which r
Return values	String	
isAtOwnNest (public) <<virtual>>		
Parameters	n/a	This returns the result of an expressi attribute **nestRow** and if the current
Return values	Bool	
updateFoodCarried (public) <<virtual>>		
Parameters	**change** : Int	This is a mutator method which upda 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 parame It then sets the following protected a • **belongsTo** to the parameter b • **strength** to the parameter initi • **pheromoneDecay** to the parar
Return values	n/a	
advanceStage (public) <<override>>		
Parameters	**nests** : Nest List **ants** : Ant List **pheromones** : Pheromone List	This method reduces the **strength** a It then checks the updated **strength**
Return values	n/a	
getBelongsTo (public)		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getStrength (public)		
Parameters	n/a	This is an accessor method which r
Return values	Int	
updateStrength (public)		
Parameters	[illegible] : Int	This is a mutator method which upda
Return values	n/a	

Class: Nest

Identifier / Data		Description
<<constructor>>		
Parameters	**startRow** : Int **startColumn** : Int **startFood** : Int	The constructor passes the parame It then sets the following protected a • **foodLevel** to the parameter st • **numberOfQueens** to 1 • **id** to the static attribute **nextNe** The constructor also increments the a new ID number.
Return values	n/a	
advanceStage (public) <<override>>		
Parameters	**nests** : Nest List **ants** : Ant List **pheromones** : Pheromone List	This method performs a number of It first checks to see if the **ants** list i exits and the nest doesn't advance. The method then initialises three in setting them all to 0. It then iterates Note that this DOESN'T check if the is a **queen** ant, the **count** variable i normal worker ant, the **count** variab incremented by 1. This is perhaps s *workerAntsBelongingToNestCou* It represents the amount of food *amount of food in the nest decrease* *ants that belong to the nest*", which regardless of their cell location in th **changeFood()** method is called, pa amount of food in the nest. The method then performs a numbe in the nest is 0 and the **antsInNest** *simulation belonging to this nest, b* incremented. If the **foodLevel** of the nest, the **antsToCull** variable is inc
Return values	n/a	

Identifier / Data		Description
		The method then performs a wider number of worker ants belonging to incremented. Within this scenario, greater than the number of worker the application from attempting to re The method then uses a For…Next removing ants from the **ants** list. Th **ants** list. The loop will not select an however, the ant selected could be be in the nest. This means that any located in a food cell or carrying foo the **numberOfQueens** variable is d If the **foodLevel** is not less than five deemed to be healthy, and therefore there is a 50% chance that the que will be a new **queen**. Combined, the new **queen** is added to the simulati are added to the **ants** list.
changeFood (public)		
Parameters	**change** : Int	This method adjusts the **foodLevel** It then checks the updated **foodLeve**
Return values	n/a	
getFoodLevel (public)		
Parameters	n/a	This is an accessor method which r
Return values	Int	

Class: Entity

Identifier / Data		Description
<<constructor>>		
Parameters	startRow : Int startColumn : Int	The con[illegible]r sets the following p • [illegible]o[illegible] the parameter startRow • [illegible]umn to the parameter start Both of these attributes are updated
Return values	n/a	
advanceStage (public) <<virtu[illegible]		
Parameters	[illegible]lest List [illegible] : Ant List pheromones : Pheromone List	This is an empty base class virtual r
Return values	n/a	
getColumn (public)		
Parameters	n/a	This is an accessor method which r
Return values	Int	
getDetails (public) <<virtual>>		
Parameters	n/a	This is a base class virtual method w
Return values	String	
getID (public)		
Parameters	n/a	This is a[illegible]ssor method which r
Return values	Int	
getRow (public)		
Parameters	n/a	This is an accessor method which r
Return values	In[illegible]	
inSameLoc[illegible] (p[illegible]blic		
Parameters	e : Entity	This returns the result of an expressi parameter e and the attribute colum 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 pa[illegible]meters to the base c It then sets the typ[illegible] attribute (from the base class
Return values	n/a	

Class: WorkerAnt

Identifier / D[illegible]		Description
<<constructo[illegible]		
Parameters	startRow : Int startColumn : Int nestInRow : Int nestInColumn : Int	The constructor passes all four parameters to the base c It then sets the typeOfAnt attribute (from the base class class) to 30.
Return values	n/a	
chooseCellToMoveTo (public) <<override>>		
Parameters	listOfNeighbours : Int List indexOfNeighbourWithStrongestPheromone : Int	This method decides how a worker ant will move based food. If it is, the method uses two IF statements to check greater than the nestRow and nestColumn respectively then updated accordingly to move the worker ant one ce Alternatively, if the worker ant is no[illegible] carrying food and th attribute is -1, this means tha[illegible] [illegible]orker ant is currently neighbouring the wo[illegible] [illegible]ich contain pheromone. I chooseRando[illegible]N[illegible]gh[illegible]() method, passing in the lis move to[illegible] [illegible]alls the changeCell() method using this If th[illegible]dexOfNeighbourWithStrongestPheromone is [illegible]h pheromone in them, which will attract it. In this case class to return the index position of the indexOfNeighbo listOfNeighbours. It then calls the changeCell() metho
Return values	n/a	
getDetails (p[illegible] [illegible]override>>		
Parameters	n/a	This method is only called from the getCellDetails meth return value from the getDetails() method in the base cl workerAnt, together with the location of its home nest.
Return values	String	

Simulation 1

	1	2	3	4	5
1	*Grid Index 0*	*Grid Index 1* **Food: [illegible]**	*Grid Index [illegible]*	*[illegible] Index 3*	*Grid Index 4*
2	*Grid In[illegible] 5*	*Grid Index 6*	*Grid Index 7*	*Grid Index 8* **Nest Ants: 5**	*Grid Index 9* **Food: 500**
3	*Grid Index 10* **Food: 500**	*Grid Index 11*	*Grid Index 12*	*Grid Index 13*	*Grid Index 14*
4	*Grid Index 15*	*Grid Index 16*	*Grid Index 17*	*Grid Index 18*	*Grid Index [illegible]*
5	*Grid Index 20*	*Grid Index [illegible]*	*[illegible] 22*	*Grid Index 23*	*Grid Index 24*

Starting N of Ne
Number c
Numb Colu
Starting A of Food i
Starting N of Food
Starting N of Ants i
Strength Pherom
Pheron Decay

Ant Simulati

Java Theory Questions

These questions refer to the **Preliminary Material** and the **Sk**
but **do not** require any additional programmin

TOTAL M S: 54

1. The Main procedure in pre-release program asks the user to enter of those is **the simulation number**.

   Curre e program assumes the user will always enter valid numer 1–4 inc e.

   (a) Explain why this program is not robust when processing user input

   (b) The code contains a prompt on line 31: Enter simulation numbe

   What error would occur during runtime if the user typed "four" whe from line 31?

   (c) How would a programmer resolve this issue?

2. Give an identifier used for:

   (a) A local variable used within the getCellReference procedure.

   (b) A class variable.

   (c) The loop counter variable used when nests are being created wit Simulation class.

   (d) A variable that is used to store the neighbouring cell indices as a l

3. This program makes use of **Encapsulation**.

   (a) Explain what encapsulation means in object-oriented programmin

   (b) Describe how this concept is used within the Cell class.

4. This program makes use of **Inheritance**.

   (a) Explain what inheritance me object-oriented programming.

   (b) Give an advantag usi this technique.

   (c) Ex h itance is used between the Entity, Ant, and Wor

5. What d structure is used to represent the grid of cells, and how ma contain in Simulation 3?

6. Describe how the advanceStage methods demonstrate polymorphism

7. Explain how the simulation demonstrates the use of abstraction by mo advantage of using this technique to simulate the behaviour of ant colo

Ant Simulati

Java Theory Questions

These questions refer to the **Preliminary Material** and the **Sk**
but **do not** require any additional programmin

TOTAL MARKS: 54

1. The Main procedure in the pre-release program asks the user to enter
of those is **the simulation number**.

Currently the program assumes the user will always enter valid numer
1–4 inclusive.

(a) Explain why this program is not robust when processing user input

....

....

....

(b) The code contains a prompt on line 31: **Enter simulation number**

What error would occur during runtime if the user typed “four” whe
from line 31?

....

....

(c) How would a programmer resolve this issue?

....

....

2. Give an identifier used for:

(a) A local variable used within the getCellReference procedure.

....

....

(b) A class variable.

....

....

(c) The loop counter variable used when nests are being created with
Simulation class.

....

....

QUESTION 2 CONTINUES OVERLEAF

Ant Simulati

Programming Tasks

These questions require you to load the **Skeleton Program** and to make

Note that any alternative or additional code changes that [...] eemed appropriate [...] ensuring that it is clear where in the [...] Program those change

The objective of this reso[...] is t[...] [...]ovide you with a selection of different que[...] questions. Some que[...] more prescriptive than others in how the task sh[...] range of le[...]s. [...] which have a similar theme may use different techni[...] options on [...] solve problems. Questions in this resource are not necessarily p

Unless specified by the question, the solutions assume valid input and therefore [...] beyond that supplied by the original pre-release material fr

Students are recommended to start with a clean copy of the pre-release code [...] questions in this resource. This will prevent modifications made for one question h[...] different question.

These questions are based on Version 2 of the Ant Simulation code released by [...] you are using Version 2 of the code from Centre Serv

Task 1

This question extends the Skeleton Program to confirm that the user has s
simulation. Modify the application to confirm that the user would like to end
when they input option 9 from the main menu. On confirmation, the progra
the simulation has ended.

What you need to do

Task 1.1

Update the **Main** method. When the user selects option 9 (quit), confirm th
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 use
- When prompted confirm to quit the simulation
- Show the program displaying a message confirming that the simulatio

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the modifications to the M
- SCREEN CAPTURE(S) showing the required tests.

Task 2

This question extends the **Ant** class and the **Nest** class to show a worker a
reaches the age of 14 (stages). Modify the **Ant** class to include a getter for
Stages to ask an ant its age. Modify the **AdvanceStage** method in the **Nes**
from the **Ants** list which are older than 14 stages. This should happen at th
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 protecte
AdvanceStage method in the **Nest** class to check if an ant is older than 14
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 protect
  modification to the **AdvanceStage** method in the **Nest** class.
- SCREEN CAPTURE(S) showing the required tests.

Ant Simulati

Programming Tasks (Extensi

Extension 1

In the standard simulation, ants will continue to follow a pheromone trail to when that food source has been exhausted. Introduce a second pheromon to the Ant class. When an ant finds that the food sou e has less than 10% lay down the new pheromone rather than the sta d r l pheromone as it ret "food source exhausted" pheromone sh ... quadruple the decay level of t help erase the path more quick.

Extens 2

In the stand simulation it is possible for the Queen ant to be culled at ra method into the Nest class which detects if a Queen has not been present promotes a normal WorkerAnt to a new Queen.

Extension 3

Introduce the concept of a soldier ant which can spray formic acid. Create moves using the normal random movement around the simulation world. A should review all of its neighbouring cells. If the soldier ant finds an ant or a different nest, it should spray formic acid towards that cell, killing all of the

Extension 4

Introduce a "message-passing" system whereby ants can leave messages heads as they move past each other. Modify the Cell class to include the c scale of 1–5 (1 being poor, 5 being highly nutritious). As ants pass each ot nest, the returning ant, carrying food, "bumps heads" with other ants to info food source. Modify the chooseCellToMoveTo method to use this informa for deciding if an ant will follow the pheromone trail.

Extension 5

Introduce the concept that a nest can have a maximum of 30 ants belongin been reached, create a new Queen ant which leaves the nest and moves a simulation to create a new nest. Two nests cann in neighbouring cells.

Extension 6

Introduce two new ... 6 and 7, to the main menu to give the user the simulation ... a CSV file.

Extension 7

Implement a dictionary data structure into the WorkerAnt class which stor source found and how much food there is. At each stage, use the strength to the food source to make a guess at how many ants have visited the foo much food is being consumed per stage, and update the dictionary. Use th chooseCellToMoveTo method as additional weighting for selecting wheth pheromone trail back to a food source.

Extension 8

Implement the A* algorithm for a worker ant to find the shortest path back t
Use Manhattan distance as the heuristic.

Extension 9

Modify the **Cell** class to allow different types of food – for example, food an
concept to the **Nest** class to monitor how much has been collected by the w
WorkerAnt class so that an ant won't collect a particular food source if ther

Extension 10

Introduce polydomous colonies which can have multiple nests. Track share
deposit food at nests within their colony rather than just at one specific nes
there is already plenty of food, the ant should go to another nest.

Extension 11

Modify the Nest class to create a new method called **shareFoodWithNear**
one nest to take food to the ants in another nest if it is running low.

Extension 12

Implement Tandem Running by implementing new **LeadAnt** and **Follower**
been found, instantiate two ants in the nest: one "Lead" and one "Follow". T
pheromone, instead moving randomly, but has double the food-carrying ca
pheromone trail, moving slowly (one cell per two stages). At each stage it s
follower is within one cell distance. If not, it waits. The follower ant has a 90
the leader's current position. It stops if already adjacent. When both ants r
ant picks up the food and takes it back to the nest.

Extension 13

Introduce the concept of "most food per unit of time" (stage). Modify the **Cel**
allowing only three ants present at any time. Introduce a new **Bridge** class v
which has a capacity of 10 ants present at any time. Add a new option 6 to t
user to place bridges into the simulation at any location except the nest. Ad
menu to allow the user to place food into cells. Demonstrate ants ignoring a
placed in a cell at a location which can't be navigated to using bridges in fa
food that is easier to get to because overall the nest gains a greater amoun

Extension 14

Introduce the concept of "unsociality" whereby a worker ant defects to a dif
more food in it and therefore offers the ant a greater chance of survival.

Extension 15

Introduce the concept of "nutritional value" in a food source. Once a unit of
to the nest, it is assessed by the ants in the nest. If it is lower than other fo
at, a **ReviewerAnt** follows the pheromone trail out to the food source, rem

Question		Suggested Solution
13		The numbers are not truly random; they are produced by a deterministic algorithm using a seed value **[1]** This means the same seed will always produce the same sequence of "random" numbers [1]
14	(a)	Dictionary lookups are O(1) on average, so finding pheromones at a cell would be faster than scanning the whole list **[1]** They provide direct access by coordinates rather than searching sequentially through a list [1]
	(b)	Dictionary keys are hashed to compute an index position. If a key is not hashable (mutable), its hash value could change, breaking dictionary lookups **[1]** Immutable types (tuples like (row, column)) are hashable and safe to use as keys **[1]**
15		In object-oriented programming, code is organised into classes which contain both data and behaviour. In a procedural program, these would instead be handled by separate functions and shared data structures, rather than being combined in a class. **[1]** Example, the **Ant** and **Nest** classes in this program store attributes like **row** and **foodLevel**, and contain methods such as **advanceStage** and **changeFood** **[1]** OOP uses encapsulation to protect data by keeping attributes private and controlling access through methods. In a procedural approach, this value might be stored in a global variable, meaning it could be accidentally changed by any part of the program. **[1]** For example, the nest's food level is stored in **foodLevel** and can only be changed using the **changeFood** method **[1]**

Task 20

Coding

- Add a suitable data structure to the WorkerAnt class which enables an ant to store its movement his
- Store the movement history of the worker ant from one stage to another. [... ark]
- Override the normal worker ant movement behaviour to backtr... ...the movement history co
- Reset the history if the worker ant finds food. [1 mark]

Teacher Notes

The LIFO structure of a stack lends its... effectively to this question; however, it could be done with a... solution uses an attribute "Fora... ...ge..." to keep track of the number of stages. This could also be do... backtracking.

Example Solu...

Modification of t... ...rkerAnt class to introduce a new data structure to store the movement history:

```
    static class WorkerAnt extends Ant {
        // CHANGE
        private Stack<int[]> movementHistory;
        private int foragingStages;
        private boolean backTracking = false;

        public WorkerAnt(int startRow, int startColumn, int nestInRow, int nestI
            super(startRow, startColumn, nestInRow, nestInColumn);
            typeOfAnt = "worker";
            foodCapacity = 30;
            movementHistory = new Stack<>();
            foragingStages = 0;
        }
        // END CHANGE
```

Modification of the ChooseCellToMoveTo meth... ...he ...kerAnt class to push movement history ont

```
        @Override
        public void c... ...ToMoveTo(ArrayList<Integer> listOfNeighbours, int 
            ...pp... 
            ...untOfFoodCarried > 0) {
            ...HANGE
                foragingStages = 0; // Covers the scenario where the ant finds f
                movementHistory.clear();
                //END CHANGE
                if (row > nestRow) {
```

```
                row -= 1;
            } else if (row < nestRow) {
                row += 1;
            }
            if (column > nestColumn) {
                column -= 1;
            } else if (column < nestColumn) {
                column += 1;
            }
        //CHANGE
        } else if (foragingStages >= 7) {
            backtrackToPreviousCell();
        } else if (indexOfNeighbourWithStrongestPheromone == -1) {
            int indexToUse = chooseRandomNeighbour(listOfNeighbours);
            r = new IntWrapper(row);
            c = new IntWrapper(column);
            changeCell(indexToUse, r, c);
            row = r.value;
            column = c.value;
            movementHistory.push(new int[]{row,column});
            foragingStages++;
        } else {
            int indexToUse = listOfNeighbours.indexOf(indexOfNeighbourWithStr
            r = new IntWrapper(row);
            c = new IntWrapper(column);
            changeCell(indexToUse, r, c);
            row = r.value;
            column = c.value;
            foragingStages = 0;        //Once it finds a pheromone trail, th
            movementHistory.clear();
        }
        //END CHANGE
    }
```

Creation of `BacktrackToPreviousCell` method which is called when ant reaches seven stages of mov

```
        //CHANGE
        } else if (foragingStages >= 7) {
            backtrackToPreviousCell();
        } else if (indexOfNeighbourWithStrongestPheromone == -1) {
            int indexToUse = chooseRandomNeighbour(listOfNeighbours);
            r = new IntWrapper(row);
            c = new IntWrapper(column);
            changeCell(indexToUse, r, c);
            row = r.value;
            column = c.value;
            movementHistory.push(new int[]{row,column});
            foragingStages++;
        } else {
            int indexToUse = listOfNeighbours.indexOf(indexOfNeighbourWithStr
            r = new IntWrapper(row);
            c = new IntWrapper(column);
            changeCell(indexToUse, r, c);
            row = r.value;
            column = c.value;
            foragingStages = 0;         //Once it finds a pheromone trail, th
            movementHistory.clear();
        }
        //END CHANGE
    // CHANGE
    public void backtrackToPreviousCell() {
        backTracking = true;
        if (!movementHistory.isEmpty()) {
            System.out.print("Ant " + id + " backtracking from " + row + ", "

            int[] previousPosition = movementHistory.pop();
            row = previousPosition[0];
            column = previousPosition[1];
            System.out.println(" to " + row + ", " + column);
            if (row == nestRow && column == nestColumn) {
                backTracking = false;
                foragingStages = 0;
            }
        }
    }
    // END CHANGE
```

Testing

- Show the program displaying backtracking route(s) of ants which have not successfully found food. [

```
> 5
Enter number of stages to advance by: 15
Ant 3 backtracked from 2, 2 to 2, 2
Ant 8 backtracked from 9, 8 to 9, 8
Ant 9 backtracked from 7, 6 to 7, 6
Ant 12 backtracked from [illegible], 6 to 6, 6
Ant 3 backtracked from 2, 2 to 2, 1
Ant 8 backtracked from 9, 8 to 8, 7
Ant 9 backtracked from 7, 6 to 8, 6
Ant 12 backtracked from 6, 6 to 7, 6
Ant 3 backtracked from 2, 1 to 1, 1
Ant 8 backtracked from 8, 7 to 7, 7
Ant 9 backtracked from 8, 6 to 7, 6
Ant 12 backtracked from 7, 6 to 7, 5
Ant 3 backtracked from 1, 1 to 2, 1
Ant 8 backtracked from 7, 7 to 6, 7
Ant 9 backtracked from 7, 6 to 6, 5
Ant 12 backtracked from 7, 5 to 7, 4
Ant 3 backtracked from 2, 1 to 3, 2
Ant 8 backtracked from 6, 7 to 7, 6
Ant 9 backtracked from 6, 5 to 6, 6
Ant 12 backtracked from 7, 4 to 6, 5
Ant 3 backtracked from 3, 2 to 4, 3
Ant 8 backtracked from 7, 6 to 6, 6
Ant 9 backtracked from 6, 6 to 5, 6
Ant 12 backtracked from 6, 5 to 7, 4
Ant 3 backtracked from 4, 3 to 3, 3
Ant 8 backtracked from 6, 6 to 6, 5
Ant 9 backtracked from 5, 6 to 4, 5
Ant 12 backtracked from 7, 4 to 6, 4
Simulation moved on 15 stages
```

Name	

ZigZag Education supporting

A Level AQA Computer Science Pap

Summer 2026

Ant Simulation

Electr Answer Document (EAD)

Instructions

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages

- Answer **all** questions
- The marks available for each question are shown in brackets

- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

Total marks:

Exam-style Questions

Answer all questions. Remember to save this document

Q		Answer
1	(a)	
	(b)	
	(c)	
2	(a)	
	(b)	
	(c)	
	(d)	
3	(a)	
	(b)	
4	(a)	
	(b)	
	(c)	
5		
6		
7		
8	(a)	
	(b)	
9	(a)	
	(b)	
10	(a)	
	(b)	
	(c)	
	(d)	
11	(a)	
	(b)	
	(c)	
	(d)	
	(e)	
	(f)	
12		
13		
14	(a)	
	(b)	
15		

Exam-style Programming Task

Answer all questions. Remember to save this document

Q	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

Ant Simulation

UML Cla