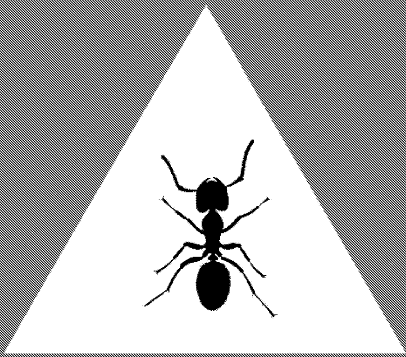
2015 specification
for the 2026 exam



Ant Simulation

PAPER 1 EXAM RESOURCE PACK 2026

for A Level AQA Computer Science

VB.NET 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 12896

Contents

Product Support from ZigZag Education ii

Terms and Conditions of Use iii

Teacher's Introduction iv

Printouts of electronic resources (for reference)

- Code Breakdown (16 pages)
- Grid Example (1 page)
- Theory Questions: Non-write-on Version (2 pages)
- Theory Questions: Write-on Version (5 pages)
- Coding Tasks (23 pages)
- Additional Tasks (Extension) (2 pages)
- Theory Questions: Mark Scheme (4 pages)*
- Coding Tasks: Mark Scheme (74 pages)*
- Electronic Answer Document (EAD) (3 pages)
- UML Class Diagram: Complete (1 page)*
- UML Class Diagram: Activity (1 page)**

** 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 Methods

Identifier / Data		Description
GetCellReference		
Parameters	**Row** : Int (passed by ref) **Column** : Int (passed by ref)	This method asks the user for the row a inputs are cast to integers and assigned passed by reference rather than by valu
Return values		
GetChoice		
Parameters	n/a	This method assigns a string input from
Return values	**Choice** : String	
DisplayMenu		
Parameters	n/a	This method displays the main program
Return values	n/a	
Main		
Parameters	n/a	This method sets up the simulation. The into list. 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 **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 at **SimulationParameters** • **StartingNumberOfNests** to th • **NumberOfRows** to the value i • **NumberOfColumns** to the valu • **StartingFoodInNest** to the val • **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 **Gri** Each cell is given a **Row** and **Colur** of this are from 1,1 to **NumberOfR** The method then calls the **SetUpA** which is the default **Row**,**Column** p 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 **NumberOfRows** + 1, or 1 to **Numb** simulation "world". If the random po present, the program loops to selec chosen, the method calls the **SetU** 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 alr added multiple times to the same c
Return values	n/a	

Identifier / Data		Description
AddFoodToNest (public)		
Parameters	**Food** : Int **Row** : Int **Column** : Int	This method uses a Foreach loop to food to a nest if a **Nest** is at the sam the nest 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 method for the cell at that index and p that cell. The skeleton program often amount of food in a cell.
Return values	n/a	
AdvanceStage (public)		
Parameters	**NumberOfStages** : Int	This method contains a For…Next loo simulation from one stage to the next. Inside the loop, the method firstly 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 me **PheromonesToDelete** list, removing list. This is because you can't remove with a Foreach loop. Once the loop has completed, the me the **AdvanceStage** method on each object called **TempCell** as a referenc investigated. The method then querie method calls the **AddFoodToNest()** together with its row and location. Thi **UpdateFoodCarried()** method is cal carrying, which will reduce the amoun
Return values	n/a	

Identifier / Data		Description
		If the ant, however, is in a cell which c a food capacity greater than zero, the capacity of the ant. AQA updated this *(derived from the Ant class)* with a foo the ant can pick up food, the method than what is available in the cell or gr UpdateFoodInCell() on the Current taken by the ant. Finally, it calls the U ant to carry back to the nest. If the ant is not in a food cell, the code its way back to the nest. If so, the me ant to reapply or generate new pheror statement then calls the ChooseCell GetIndicesOfNeighbours() and Ge controls whether the ant is moving ra The final task in the method is to itera each **Nest** in the **Nests** list.
GetAreaDetails (public)		
Parameters	**StartRow** : Int **StartColumn** : Int **EndRow** : Int **EndColumn** : Int	Uses nested iteration to enumerate t the top left and bottom right corners The method uses concatenation to b in the **Details** string represents a si called TempCell as a reference of th and uses the **GetNestInCell()**, **GetN** and **GetAmountOfFood()** helper me Once the loop has completed, the De
Return values	**Details** : String	

Identifier / Data		Description
GetCellDetails (public)		
Parameters	**Row** : Int **Column** : Int	This method is used to build up a more d method instantiates a local cell object calle location being investigated. It first calls the copy of the Nest in that cell. If there is a ne that cell to find out how many food units ar The method then uses the same technique passing in the **CurrentCell** object. If there many ants are in that cell, adding that valu **GetNumberOfPheromonesInCell()** meth they are. Finally, the method adds some carriage re returns the string.
Return values	**Details** : String	
GetDetails (public)		
Parameters	n/a	Uses nested iteration to enumerate throu The method uses concatenation to build **Details** string represents a single cell in t **TempCell** as a reference of the cell in the the **GetNestInCell()**, **GetNumberOfAnts** **GetAmountOfFood()** helper methods to Once the loop has completed, the **Details**
Return values	**Details** : String	
GetIndex (private)		
Parameters	**Row** : Int **Column** : Int	Uses the **Row** and **Column** parameters t
Return values	Int	

Identifier / Data		Description
GetIndexOfNeighbourWithStrongestPheromone (private)		
Parameters	**Row** : Int **Column** : Int	This method firstly instantiates the fol • **StrongestPheromone** to 0 • **IndexOfStrongestPheromone** Then uses a Foreach loop to iterate cell at position **Row**, **Column**. This is method. If a neighbour is not valid (fo is represented as -1. In the Foreach lo the strength of the pheromone in that so, the method updates the **IndexOfS** what the new **StrongestPheromone** After the loop has completed, the **Ind** contain any pheromones, the **IndexO** continue moving randomly.
Return values	**IndexOfStrongestPheromone** : Int	
GetIndicesOfNeighbours (private)		
Parameters	**Row** : Int **Column** : Int	This method builds up and returns a li **Grid** indices of the neighbouring cells at the position **Row**, **Column**. The me nested iteration to look at the cells in a around the cell given at position **Row**, For example, if **Row** was 3 and **Colu** in a standard 5×5 simulation "world", t method would return a list containing [5,6,7,10,-1,12,15,16,17]. If a cell is outside the bounds of the si "world", or if the cell being investigate the index is given as -1.
Return values	**ListOfNeighbours** : Int List	
GetNestInCell (public)		
Parameters	**C** : Cell	This method iterates through the **Nes** parameter **C**. If they match, the **Nest** the method gives a null return.
Return values	**N** : Nest	

Identifier / Data		Description
GetNumberOfAntsInCell (public)		
Parameters	**C**: Cell	This method initialises an integer vari the location of the **Ant** with the locatio incre[illegible]. After the loop, **Count** is
Return values	**Count** : Int	
GetNumberOfPheromonesInCell (public)		
Parameters	**C**: Cell	This method initialises an integer vari comparing the location of the **Pherom** **Count** variable is incremented. After t
Return values	**Count** : Int	
GetStrongestPheromoneInCell (private)		
Parameters	**C**: Cell	This method initialises an integer vari list comparing the location of the **Phe** if the **Count** variable and the strength the variable **Strongest**, the value in t **Pheromone**. After the loop, **Stronge**
Return values	**Strongest** : Int	
SetUpANestAt (public)		
Parameters	**Row** : Int **Column** : Int	This method first adds a new **Nest** ob **Nests** list. It then adds a new **Queen** **Ants** list. Finally, it uses a For…Next has a lower bound of 2 and the upper **StartingAntsInNest** must also incluc
Return values	n/a	
UpdateAntsPheromoneInCell (public)		
Parameters	**A** : Ant	This method iterates through the **Phe** parameter **A**, and it has the same ID a the **UpdateStrength()** method on the parameter, and then exits the method without finding a match, it means that original path; therefore, the method in the **Pheromone** list.
Return values	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 Get 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 reference) **ColumnToChange** : Int (passed by reference)	This method updates the position of th on the New[illegible]Indicator parameter. the ri[illegible]ws the location of neighbo arou[illegible]nt. The **NewCellIndicato** [illegible]x position. If the parameter is grea then the new direction must be downw therefore, the method increments the **RowToChange** parameter to move th downwards. If the **NewCellIndicator** 3, the new direction must be upwards; the method decrements the **RowToC** If the **NewCellIndicator** is 0, 3 or 6, t the **ColumnToChange** parameter to direction must be right; therefore, the r the ant right. A combination of these s table shown above. **RowToChange** and **ColumnToCha** be returned.
Return values	n/a	
ChooseCellToMoveTo (public) <<virtual>>		
Parameters	**ListOfNeighbours** : Int List **IndexOfNeighbourWithStrongestPheromone** : int	This is an empty base class virtual me
Return values	n/a	
ChooseRandomNeighbour (protected)		
Parameters	**List[illegible]** : 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 conta location which is outside the bounds
Return values		

Identifier / Data		Description
GetDetails (public) <<override>>		
Parameters	n/a	This method is only called from the
Return values	String	comprises the return value from the the **Ant**, the **TypeOfAnt**, and how m method in the base class, however, j
GetFoodCapacity (public) <<override>>		
Parameters	n/a	This is an accessor method which re
Return values	Int	
GetFoodCarried (public) <<virtual>>		
Parameters	n/a	This is an accessor method which re
Return values	Int	
GetNestColumn (public) <<virtual>>		
Parameters	n/a	This is an accessor method which re
Return values	Int	
GetNestRow (public) <<virtual>>		
Parameters	n/a	This is an accessor method which re
Return values	Int	
GetTypeOfAnt (public) <<virtual>>		
Parameters	n/a	This is an accessor method which re
Return values	String	
IsAtOwnNest (public) <<virtual>>		
Parameters	n/a	This returns the result of an expressio attribute **NestRow** and if the current
Return values	Bool	
UpdateFoodCarried (public) <<virtual>>		
Parameters	**Change** : Int	This is a mutator method which updat 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** 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	C[illegible] : I[illegible]	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 S • **NumberOfQueens** to 1 • **ID** to the static attribute **NextN** The constructor also increments th 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 exits and the nest doesn't advance. The method then initialises three in setting them all to 0. It then iterates Note that it DOESN'T check if the a **Queen** ant, the **Count** variable is i normal worker ant, the **Count** varia incremented by 1. This is perhaps s *Worker.AntsBelongingToNestCo* but 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, bu* incremented. If the **FoodLevel** of th this nest, the **AntsToCull** variable is
Return values	n/a	

Identifier / Data		Description
		The method then performs a wider number of worker ants belonging to incremented. Within this scenario, t greater than the number of worker a 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 fiv is deemed to be healthy, and theref there is a 50% chance that the Que will be a new **Queen**. Combined, th a new **Queen** is added to the simula ants are added to the **Ants** list.
ChangeFood (public)		
Parameters	**Change** : Int	This method adjusts the **FoodLeve** It then checks the updated **FoodLe** back to 0.
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 constructor sets the following p • Row to the parameter **StartRo** • Column to the parameter **Star** Both of these attributes are update
Return values	n/a	
AdvanceStage (public) <<virtual>>		
Parameters	**Nests** : Nest List **Ants** : Ant List **Pheromones** : Pheromone List	This is an empty base class virtual
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 an accessor method which r
Return values	Int	
GetRow (public)		
Parameters	n/a	This is an accessor method which r
Return values	Int	
InSameLocation (public)		
Parameters	**E** : Entity	This returns the result of an expressi parameter **E** and the attribute **Colum** entities are in the same place).
Return values	Bool	

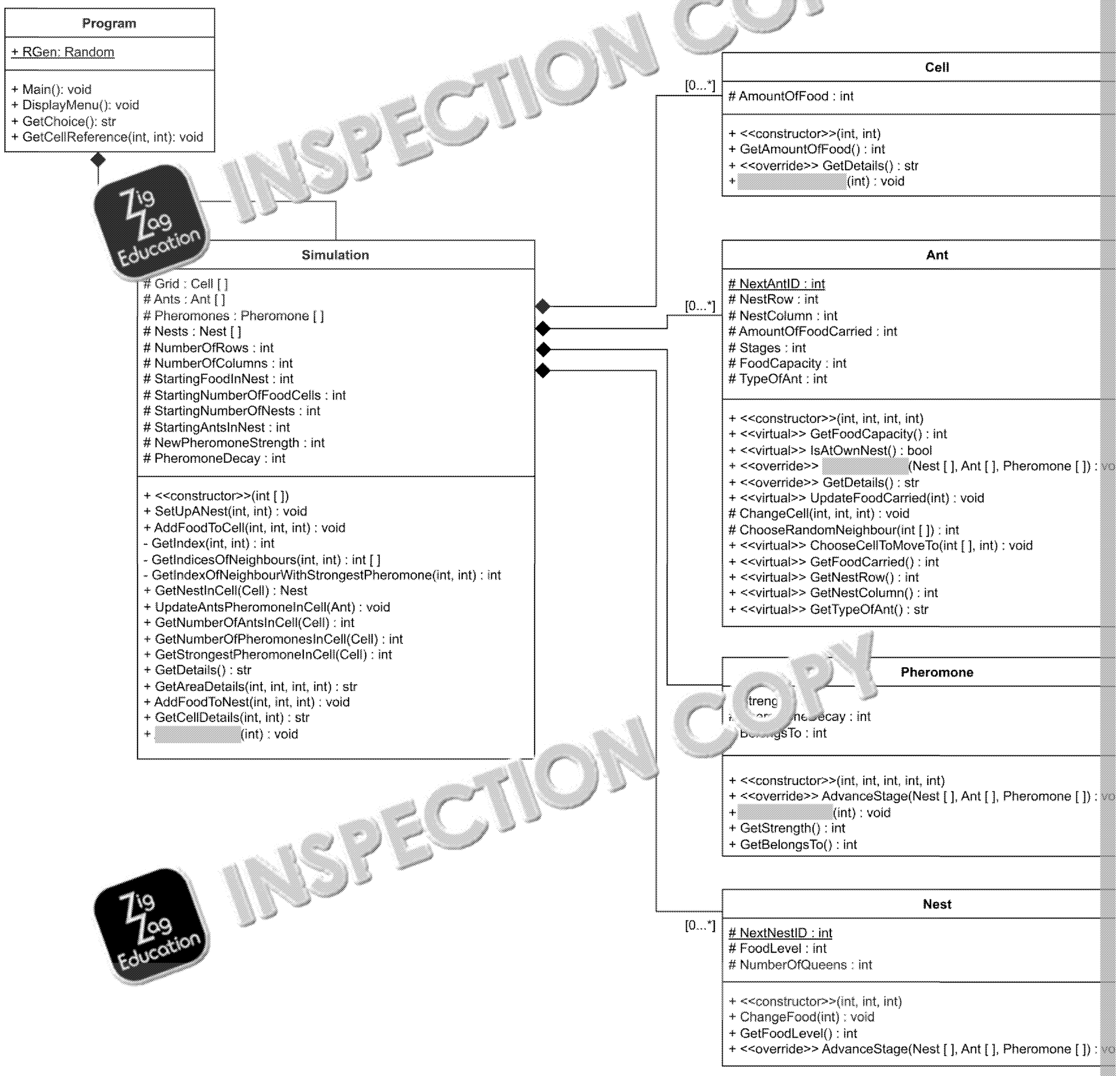
Class: Queen

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int **NestInRow** : Int **NestInColumn** : Int	The constructor passes all four para It then sets the **TypeOfAnt** attribute
Return values	n/a	

Class: WorkerAnt

Identifier / Data		Description
<<constructor>>		
Parameters	**StartRow** : Int **StartColumn** : Int **NestInRow** : Int **NestInColumn** : Int	The constructor passes all four para It then sets the **TypeOfAnt** attribute attribute (from the base class) to 30
Return values	n/a	
ChooseCellToMoveTo (public) <<override>>		
Parameters	**ListOfNeighbours** : Int List **IndexOfNeighbourWithStrongestPheromone** : Int	This method decides how a worker ant is carrying any food. If it is, the **Column** of the worker ant are less respectively. The **Row** and **Column** to move the worker ant one cell per Alternatively, if the worker ant is not **IndexOfNeighbourWithStrongest** currently searching for food and the con pheromone. In this case the passing in the **ListOfNeighbours** p calls the **ChangeCell()** method usin If the **IndexOfNeighbourWithStron** next to a cell or cells with pheromon the **IndexOf()** method in the built-in **IndexOfNeighbourWithStrongest** the **ChangeCell()** method using this
Return values	n/a	
GetDetails (public) <<override>>		
Parameters	n/a	This method is only called from the which comprises the return value fr **AmountOfFoodCarried** of the Wo
Return values	String	

Ant Simulation



Simulation 1

	1	2	3	4	5
1	*Grid Index 0*	*Grid Index 1* **Food:** [illegible]	*Grid Index* [illegible]	*Grid Index 3*	*Grid Index 4*
2	*Grid Index 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 2*[illegible]	*Grid Index 22*	*Grid Index 23*	*Grid Index 24*

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

Ant Simulati

VB 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
One 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 inpu

(b) The code contains a prompt on line 12: **Enter simulation numbe**

What error would occur during runtime if the user typed "four" whe
from line 12?

(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 means in object-oriented programming.

(b) Give an advantage of using this technique.

(c) Explain how inheritance is used between the Entity, Ant, and Wor

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

6 Describe how the **AdvanceStage** methods demonstrate polymorphis

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

Ant Simulati

VB 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 program asks the user to enter
One of those is **the simulation number**.

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

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

....

....

....

(b) The code contains a prompt on line 12: **Enter simulation number**

What error would occur during runtime if the user typed "four" whe
from line 12?

....

....

(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 q... more prescriptive than others in how the task sh... range of l... 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 A... 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 user
- 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 protecte 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 source has less than 10%
lay down the new pheromone rather than the standard pheromone as it ret
"food source exhausted" pheromone should quadruple the decay level of th
help erase the path more quickly.

Extension 2

In the standard 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 a
moves using the normal random movement around the simulation world. At
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 a

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 cannot be in neighbouring cells.

Extension 6

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

Extension 7

Implement a dictionary data structure into the **WorkerAnt** class which store
source found and how much food there is. At each stage, use the strength o
to the food source to make a guess at how many ants have visited the food
much food is being consumed per stage, and update the dictionary. Use thi
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 v
WorkerAnt class so that an ant won't collect a particular food source if the
nest.

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 then it should go to another nest.

Extension 11

Modify the **Nest** class to create a new method called **ShareFoodWithNea**
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 re
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 **Ce**
allowing only three ants present at any time. Introduce a new **Bridge** class
cell which has a capacity of 10 ants present at any time. Add a new option
the user to place bridges into the simulation at any location except the nest
main menu to allow the user to place food into cells. Demonstrate ants ign
being placed in a cell at a location which can't be navigated to using bridge
amount of food that is easier to get to because overall the nest gains a gre
time.

Extension 14

Introduce the concept of "[illegible]" whereby a worker ant defects to a di
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

Preview of Questions Ends Here

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 for 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. [1 mark]
- Override the normal worker ant movement behaviour to backtrack through the movement history cor
- Reset the history if the worker ant finds food. [1 mark]

Teacher Notes

The LIFO structure of a stack lends itself very effectively to this question; however, it could be done with a

Example Solution

Modification of the WorkerAnt class to introduce a new data structure to store the movement history:

```
    Class WorkerAnt
        Inherits Ant

        'CHANGE
        Private MovementHistory As Stack(Of Tuple(Of Integer, Integer))
        Private ForagingStages As Integer
        Private BackTracking As Boolean = False

        Public Sub New(StartRow As Integer, StartColumn As Integer, NestInRow As In
            MyBase.New(StartRow, StartColumn, NestInRow, NestInColumn)
            TypeOfAnt = "worker"
            FoodCapacity = 30
            MovementHistory = New Stack(Of Tuple(Of Integer, Integer))()
            ForagingStages = 0
        End Sub
        'END CHANGE
```

Modification of the ChooseCellToMoveTo method in the WorkerAnt class to push movement history ont

```
        Public Overrides Sub ChooseCellToMoveTo(ListOfNeighbours As List(Of Integer)
            If AmountOfFoodCarried > 0 Then
                'CHANGE
                ForagingStages = 0
                MovementHistory.Clear()
                If Row > NestRow Then
                    Row -= 1
                ElseIf Row < NestRow Then
                    Row += 1
                End If
                If Column > NestColumn Then
                    Column -= 1
                ElseIf Column < NestColumn Then
                    Column += 1
                End If
```

```
        ElseIf ForagingStages >= 7 Then
            BacktrackToPreviousCell()
        ElseIf IndexOfNeighbourWithStrongestPheromone = -1 Then
            Dim IndexToUse As Integer = ChooseRandomNeighbour(ListOfNeighbours)
            ChangeCell(IndexToUse, Row, Column)
            Console.WriteLine($"ForagingStages for this Ant: {ForagingStages}")
            MovementHistory.Push(Tuple.Create(Row, Col... he ant is foragin
            ForagingStages += 1
        Else
            Dim IndexToUse As Integer = ...eighbours.IndexOf(IndexOfNeighbo
            ChangeCell(IndexToUs... Column)
            ForagingStages = ... Once ... finds a pheromone trail, the ant is no
            MovementHi...
        End If
        'END CHAN...
    End Sub
```

Creation of Bac...ToPreviousCell method which is called when ant reaches seven stages of mov

```
            'CHANGE
        ElseIf ForagingStages >= 7 Then
            BacktrackToPreviousCell()
            'END CHANGE
        ElseIf IndexOfNeighbourWithStrongestPheromone = -1 Then
            Dim IndexToUse = ChooseRandomNeighbour(ListOfNeighbours)
            ChangeCell(IndexToUse, Row, Column)
        Else
            Dim IndexToUse = ListOfNeighbours.IndexOf(IndexOfNeighbourWithStron
            ChangeCell(IndexToUse, Row, Column)
        End If
    End Sub

    'CHANGE
    Public Sub BacktrackToPreviousCell()
        BackTracking = True
        If MovementHistory.Count > 0 Then
            Console.Write($"Ant {ID} ...ed from {Row}, {Column}")
            Dim PreviousPositio... Of Integer, Integer) = MovementHistory
            Row = Previous...on.Item1
            Column = ...Position.Item2
            ...ine($" to {Row}, {Column}")
            ... Ro... NestRow AndAlso Column = NestColumn Then
                BackTracking = False
                ForagingStages = 0
            End If
        End If
    End Sub
    'END CHANGE
```

Testing

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

```
1. Display overall details
2. Display area details
3. Inspect cell
4. Advance one stage
5. Advance X stages
9. Quit

> 5
Enter number of stages to advance by: 15
Ant 3 backtracked from 1, 3 to 1, 3
Ant 8 backtracked from 1, 5 to 1, 5
Ant 10 backtracked from 10, 3 to 10, 3
Ant 3 backtracked from 1, 3 to 1, 4
Ant 8 backtracked from 1, 5 to 2, 5
Ant 10 backtracked from 10, 3 to 9, 3
Ant 3 backtracked from 1, 4 to 1, 5
Ant 8 backtracked from 2, 5 to 1, 5
Ant 10 backtracked from 9, 3 to 8, 4
Ant 3 backtracked from 1, 5 to 2, 5
Ant 8 backtracked from 1, 5 to 2, 6
Ant 10 backtracked from 8, 4 to 7, 5
Ant 3 backtracked from 2, 5 to 3, 5
Ant 8 backtracked from 2, 6 to 3, 5
Ant 10 backtracked from 7, 5 to 6, 6
Ant 3 backtracked from 3, 5 to 2, 5
Ant 8 backtracked from 3, 5 to 4, 5
Ant 10 backtracked from 6, 6 to [illegible], 5
Ant 3 backtracked from 2, [illegible] to [illegible], 4
Ant 8 backtracked from [illegible], [illegible] to 5, 4
Ant 10 backtracked from 5, 5 to 5, 4
Simulation moved on 15 stages
```

Preview of Answers Ends Here

Name	

ZigZag Education supporting

A Level AQA Computer Science Paper 1

Summer 2026

Ant Simulation

Electronic 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 regularly.

Q		Answer	*Mark (leave blank)*
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 Tasks

Answer all questions. Remember to save this document regularly.

Q	Answer	*Mark* *(leave blank)*
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

