A sign with a black ant

AI-generated content may be incorrect. **Ant** **Simulation** A sign with a black ant

AI-generated content may be incorrect.

**C# Theory Questions (Mark Scheme)**

| **Question** | | **Suggested Solution** | **Total Marks** | **Marking Guidance** |
| --- | --- | --- | --- | --- |
| **1** | (a) | The program does not make use of any validation techniques **[1]**  Invalid input will cause the program to either crash or produce incorrect output **[1]** | 2 marks |  |
| (b) | Index out of range error (due to trying to access indices of SimulationParameters when it is empty in the constructor of the Simulation class) | 1 mark | **R:** any mention of type coercion failure as the input is never converted to a string  Alternatives:   Index out of range error   Index error when accessing the list/array   Out of bounds error when using the list of parameters |
| (c) | Ensure the input is validated to be an integer in the range 1–4 | 1 mark | Student may discuss **exception handling** in response, but the outcome must be that the simulation number eventually is in this range. |
| **2** | (a) | Row // Column | 1 mark |  |
| (b) | NextAntID // NextNestID | 1 mark |  |
| (c) | Count // N | 1 mark |  |
| (d) | ListOfNeighbours | 1 mark |  |
| **3** | (a) | Grouping data **[1]** (attributes) and the methods that operate **[1]** on that data together within a class, and restricting direct access to the internal state of an object **[1]** | 2 marks | **A:** Any 2 from the 3 points | |
| (b) | Direct access to the instance variable AmountOfFood is restricted as it is protected **[1]**  Getters and Setters have been provided to access or modify this instance variable **[1]** | 2 marks |  | |

| **Question** | | **Suggested Solution** | | **Total Marks** | | **Marking Guidance** | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4** | (a) | Inheritance is an object-oriented programming concept where a new class (called a subclass or child class) inherits the attributes and methods of an existing class (the superclass or parent class) and can add or override functionality. | 2 marks | | 1 mark for mentioning child/parent class  1 mark for additional functionality | |
| (b) | It allows code to be reused, so common attributes and methods don’t need to be rewritten in each class. | 1 mark | |  | |
| (c) | The Ant class inherits from Entity, so it gains the attributes Row, Column, and methods like GetRow and GetColumn **[1]**  The WorkerAnt class then inherits from Ant, reusing this functionality and adding new behaviour such as carrying food and choosing where to move **[1]** | 2 marks | |  | |
| **5** |  | A one-dimensional array or list structure is used to store all the Cell objects that make up  the grid **[1]**  In Simulation 3, the grid has 10 rows and 10 columns, giving a total of 100 elements **[1]** | 2 marks | |  | |
| **6** |  | The AdvanceStage method is defined in the parent class Entity and then overridden in several subclasses such as Ant, Nest, and Pheromone **[1]**  Each subclass provides its own version of the method, so when AdvanceStage is called, the program automatically runs the correct version for that particular object type **[1]** | 2 marks | | 1 mark for identifying that the method is overridden in subclasses  1 mark for explaining that the version executed depends on the object’s class (showing polymorphic behaviour) | | |
| **7** |  | The program models complex ant-colony behaviour using simplified objects (Ant, Nest, Pheromone, Cell) and basic rules such as movement and food collection **[1]**  Unnecessary biological details are hidden, while only essential features are represented **[1]**  This abstraction makes the problem easier to understand and simulate **[1]** | 3 marks | | 1 mark for life being too complex to  simulate fully  1 mark for hiding details while maintaining some essential features  1 mark for this allowing a simplified simulation to run | | |
| **8** | (a) | It does this by using a formula such as:  (Row - 1) \* NumberOfColumns + (Column - 1) **[1]**  This calculation finds the position of the cell in the list based on how many columns come before it in previous rows **[1]** | 2 marks | |  | | |
| (b) | 9 | 1 mark | |  | | |

| **Question** | | **Suggested Solution** | | **Total Marks** | | **Marking Guidance** | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | (a) | It can only be accessed and used from within the Simulation class itself, and cannot be called directly from outside that class or by other classes/objects. | 1 mark | |  | |
| (b) | When called with Row = 1 and Column = 1, the method would return a list of nine values where only the neighbours to the right (1, 2), below (2, 1), and diagonally below-right (2, 2) are valid  All other directions are outside the grid and therefore returned as −1  For a 5×5 grid, the list would be:  [-1, -1, -1, -1, -1, 1, -1, 5, 6] | 2 marks | | 1 mark if the student showcased that the only valid neighbouring cells are right, below, and diagonal below right  1 mark for the correctly returned list | |
| **10** | (a) | The worker ant moves one step directly towards its own nest, adjusting its row and column positions to return home with the food. | 1 mark | |  | |
| (b) | The worker ant moves to a random neighbouring cell, as there are no pheromone trails to follow. | 1 mark | |  | |
| (c) | The worker ant moves to the neighbouring cell with the strongest pheromone trail, following the scent laid by other ants. | 1 mark | |  | |
| (d) | The queen ant does not move. She remains in the same cell as the nest throughout the simulation. The QueenAnt does not override the ChooseCellToMoveTo method from the Ant class. In that method, there are no movement instructions. | 1 mark | | Either point given about not moving or why not moving awardable | |
| **11** | (a) | 20 units **[1]** | 1 mark | | (1 \* 10 + 5 \* 2) | |
| (b) | The Nest class has a changeFood method that sets the FoodLevel to 0 if it goes negative after decrementing the change | 1 mark | |  | |
| (c) | 2 ants // 1 ant culled for 7 < 10, and 1 ant culled for 7 < 50 | 1 mark | |  | |
| (d) | 0 ants // 70 is not less than 10, and 70 is also not less than 50, so no ants are culled | 1 mark | |  | |
| (e) | For the number of queens in the nest **[1]**  50% chance of creating an ant at all **[1]**  If it is decided that an ant is created:   2% chance / 98% chance respectively of a new queen / worker to be created **[1]**   (in the location of the current nest) **[1]** | 3 marks | | 1 mark for each point given  MAX 3 from 4 | |
| (f) | Removing an element from a list at a random index takes O(n) time **[1]**  Because all the elements after that index must be shifted one position to fill the gap **[1]** | 2 marks | |  | |
| **12** |  | The ChooseCellToMoveTo method is abstract, meaning it is defined in the Ant class but has no implementation **[1]**  Subclasses such as WorkerAnt provide their own version of the method, defining how that type of ant chooses where to move **[1]** | 2 marks | |  | | |

| **Question** | | **Suggested Solution** | | **Total Marks** | | **Marking Guidance** |
| --- | --- | --- | --- | --- | --- | --- |
| **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]** | 2 marks | |  | |
| **14** | (a) | Dictionary lookups are O(1) on average, so finding pheromones in 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]** | 1 mark | | 1 mark for either point | |
| (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]** | 2 marks | | 1 mark for stating that non-hashable keys could change the index when applying the hashing algorithm to find the location  1 mark for stating that dictionaries require hashable keys | |
| **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]** | 4 marks | | 4 marks: two clear, accurate differences, each explained and supported by a relevant program example  3 marks: two differences identified, one fully explained with example  2 marks: two basic differences stated with little or no explanation  1 mark: one partially correct difference | |