A-level
COMPUTER SCIENCE

Object Oriented programming

Aggregation Question & Specimen Section C & D

Published: Autumn 2017

A-level Computer Science

Name:

Class:

Object Oriented Programming

Aggregation Question

Author: AQA

Date:

Time:	**15**
Marks:	**11**

Comments:

Q1.

(a) In object-oriented programming, what is meant by aggregation?

....

....

(1)

(b) An object-oriented program is required to handle details of items of furniture that are for sale. The furniture sold includes dining suites. A dining suite consists of a table and a number of chairs.

Some fields required for the suites are
TableType
ChairType
NumberOfChairs

A method required for the suites is
DisplayDetails

Some fields required for the tables are
TableType
Size
Colour

Some fields required for the chairs are
ChairType
Colour

(i) Draw **a class diagram** of these classes, Suite, Table and Chair.

(2)

(ii) Write class definitions for Chair, Table and Suite.

....

....

....

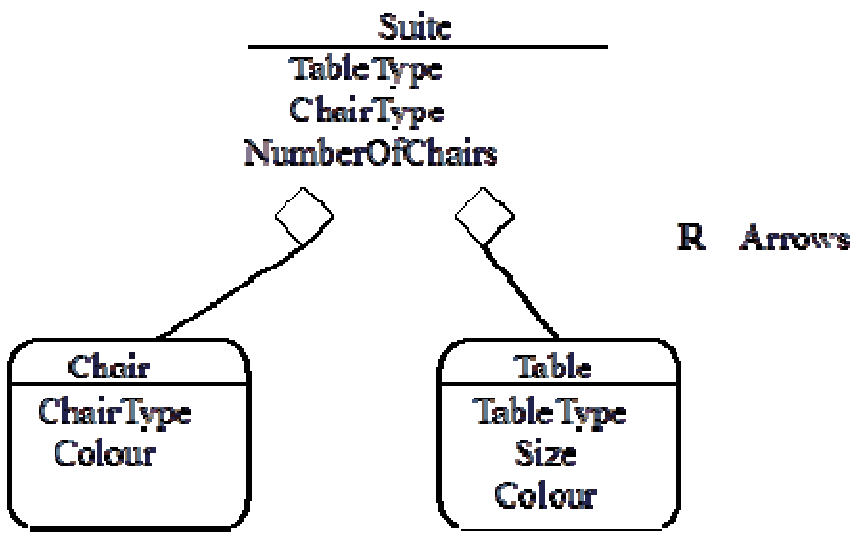
(8)
(Total 11 marks)

M1.

(a) An object that contains other objects;
A a class containing other classes;

1

(b) (i)



1 mark for class entries
1 mark for connections
A circles or diamonds, filled or not

2

(ii)

```
Chair = class;
   Private;
      ChairType : string/text    A integer       }
      Colour : string/text    A integer/color    }  ;

Table = class;
   Private
      TableType : string/text A integer      }  ;
      Size : string/text A integer           }
      Colour : string/text A integer/color   }

Suite = class;
  (Public)
     (procedure) DisplayDetails;
   Private              }
      TableType : Table }
      ChairType : Chair }  ;
      NumberOfChairs : integer;
```

A any sensible syntax
R implied inheritance

Max 8

[11]

A-level Computer Science

Object Oriented Programming

Name:

Specimen Section C

Class:

Author: AQA

Date:

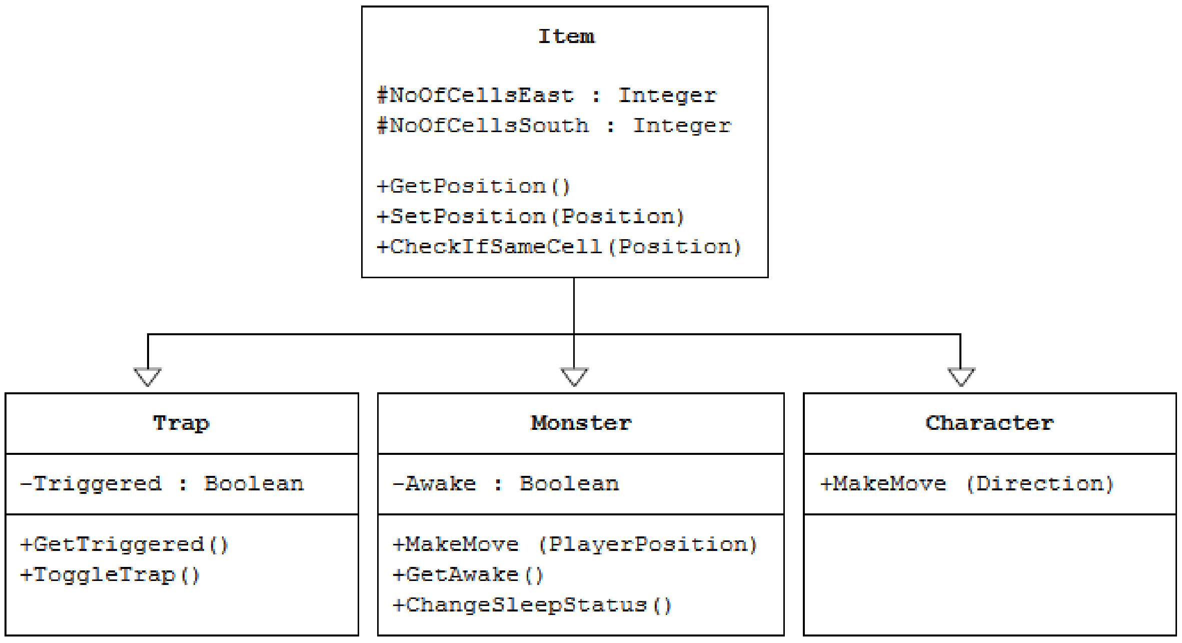
Time: **15**

Marks: **12**

Comments:

Q1.

The class diagram below is an attempt to represent the relationships between some of the classes in the MONSTER! Game.



(a) Explain what errors have been made in the class diagram. **(2)**

(b) Give an example of instantiation from the **Skeleton Program**. **(1)**

(c) State the name of an identifier for an array variable. **(1)**

(d) State the name of an identifier for a subclass. **(1)**

(e) State the name of an identifier for a variable that is used to store a whole number.

(1)

(f) State the name of an identifier for a class that uses composition.

(1)

(g) Look at the `GetNewRandomPosition` subroutine in the `Game` class in the **Skeleton Program**.

Explain why the generation of a random position needs to be inside a repetition structure.

(1)

(h) Look at the `Game` class in the **Skeleton Program**.

Why has a named constant been used instead of the numeric value `5`?

(2)

(i) Describe the changes that would need to be made to the `Game` class to add a third trap to the cavern. The third trap should have exactly the same functionality as the other two traps. You do **not** need to describe the changes that would need to be made to the `SetUpGame` subroutine.

(2)
(Total 12 marks)

M1.

(a) **All marks AO2 (analyse)**

1 mark: The arrow should be pointing towards the base class;
1 mark: There is no class called `Monster` / / it should say `Enemy`, not `Monster;`

2

(b) **Mark is for AO2 (apply)**

VB.Net

```
Dim MyGame As New Game(False) / /
Dim MyGame As New Game(True) / /
Private Player As New Character / /
Private Cavern As New Grid(NSDistance, WEDistance) / /
Private Monster As New Enemy / /
Private Flask As New Item / /
Private Trap1 As New Trap / /
Private Trap2 As New Trap;
```

R If any additional code
R If spelt incorrectly
I Case

1

(c) **Mark is for AO2 (apply)**

VB.Net
`CavernState;`

R If any additional code
R If spelt incorrectly
I Case

1

(d) **Mark is for AO2 (apply)**

`Trap / / Character / / Enemy;`

A `SleepyEnemy`
R If any additional code
R If spelt incorrectly
I Case

1

(e) **Mark is for AO2 (apply)**

`Choice / / NoOfCellsEast / / NoOfCellsSouth / / Count / / NSDistance / / WEDistance / / Count1 / / Count2;`

R If any additional code
R If spelt incorrectly
I Case

1

(f) **Mark is for AO2 (apply)**

`Game;`

R If any additional code
R If spelt incorrectly
I Case

1

(g) **Mark is for AO2 (analyse)**

So that a position of (0,0) is rejected / / so that the item can't be in the player's starting position;

1

(h) **Marks are for AO1 (understanding)**

Makes the program code easier to understand;
Makes it easier to update the program;
Makes it easier to change the size of the cavern (in the game);
Max 2 points from the list above

2

(i) **Marks are for AO2 (analyse)**

1 mark: Create a new object (`Trap3`) of class `Trap;`
1 mark: Change the (3rd) `If` statement in the `PlayGame` subroutine by adding conditions to check if the player is in the same cell as `Trap3` and that `Trap3` has not been triggered already;

2

[12]

A-level Computer Science

Name:

Class:

Object Oriented Programming

Specimen Section D

Author: AQA

Date: Time: **60**

Marks: **35**

Comments:

Q1.

(a) This question refers to the subroutines `CheckValidMove` and `Play` in the `Game` class.

The **Skeleton Program** currently does not make all the checks needed to ensure that the move entered by a player is an allowed move. It should not be possible to make a move that takes a player outside the 7 × 5 cavern grid.

The **Skeleton Program** needs to be adapted so that it prevents a player from moving west if they are at the western end of the cavern.

The subroutine `CheckValidMove` needs to be adapted so that it returns a value of FALSE if a player attempts to move west when they are at the western end of the cavern.

The subroutine `Play` needs to be adapted so that it displays an error message to the user if an illegal move is entered. The message should state `"That is not a valid move, please try again"`.

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(3)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `Play`.

(2)

(iii) SCREEN CAPTURE(S) for a test run showing a player trying to move west when they are at the western end of the cave.

(1)

(b) This question will extend the functionality of the game.

The game is to be altered so that there is a new type of enemy: a sleepy enemy. A sleepy enemy is exactly the same as a normal enemy, except that after making four moves it falls asleep again.

Task 1
Create a new class called `SleepyEnemy` that inherits from the `Enemy` class.

Task 2

Create a new integer attribute in the `SleepyEnemy` class called `MovesTillSleep`.

Task 3
Create a new public subroutine in the `SleepyEnemy` class called `ChangeSleepStatus`. This subroutine should override the `ChangeSleepStatus` subroutine from the `Enemy` class. The value of `MovesTillSleep` should be set to 4 in this subroutine.

Task 4
Create a new public subroutine in the `SleepyEnemy` class called `MakeMove`. This subroutine should override the `MakeMove` subroutine from the `Enemy` class. When called this subroutine should reduce the value of `MovesTillSleep` by 1 and then send the monster to sleep if `MovesTillSleep` has become equal to 0.

Task 5
Modify the `Game` class so that the Monster object is of type `SleepyEnemy` (instead of `Enemy`) .

Task 6
Check that the changes you have made work by conducting the following test:

- play the training game
- move east
- move east
- move south.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new `SleepyEnemy class`.

(8)

(ii) SCREEN CAPTURE(S) showing the requested test.

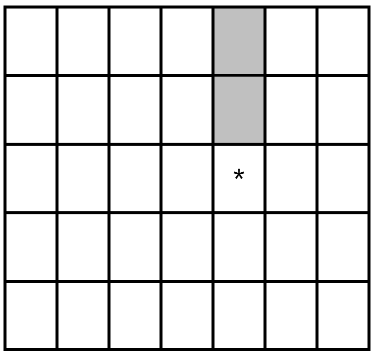
(2)

(c) This question refers to the `Game` and `Character` classes and will extend the functionality of the game.

The game should be altered so that once per game the player can shoot an arrow instead of making a move in the cavern. The arrow travels in a straight line, in a direction of the player's choice, from the cell the player is in to the edge of the cavern. If the arrow hits the monster then the player wins the game and a message saying that they have shot the monster should be displayed.

For this question you are **only** required to extend the program so that it checks if the monster is hit by the arrow when the user chooses to shoot an arrow northwards. However, the user should be able to select any of the four possible directions.

In the diagram below, the two shaded cells show the cells which, if the monster is in one of them, would result in the player winning the game, as long as the player is in the cell five to the east and three to the south and chooses to shoot an arrow northwards.



Task 1
Modify the `DisplayMoveOptions` subroutine in the `Game` class so that the option to enter A to shoot an arrow is added to the menu.

Task 2
Create a new Boolean attribute called `HasArrow` in the `Character` class.

The value of `HasArrow` should be set to `True` when a new object of class `Character` is instantiated.

Task 3
Create a new public subroutine called `GetHasArrow` in the `Character` class that returns the value of the `HasArrow` attribute to the calling routine.

Task 4
Modify the `CheckValidMove` subroutine in the `Game` class so that:

- it is a valid move if A is selected and the player does have an arrow
- it is not a valid move if A is selected and the player does not have an arrow.

Task 5
Create a new public subroutine called `GetArrowDirection` in the `Character` class.

This subroutine should return a character to the calling routine.

The user should be asked in which direction they would like to shoot an arrow (N, S, E or W) and the value entered by the user should be returned to the calling routine.

If an invalid direction is entered then the user should be repeatedly asked to enter a new direction, until a valid direction is entered.

The value of `HasArrow` should then be changed to FALSE.

Task 6
Modify the `Play` subroutine in the `Game` class so that if the move chosen by the user is not M it then checks if the move chosen is A.

If the move chosen was A, then there should be a call to the player's `GetArrowDirection` subroutine. If the user chooses a direction of N then the program should check to see if the monster is in one of the squares directly north of the player's current position. If it is then a message saying `"You have shot the monster and it cannot stop you finding the flask"` should be displayed. The value of `FlaskFound` should then be set to TRUE.

After the arrow has been shot, if the monster is still alive and awake, it is now the monster's turn to move, the player should remain in the same cell as they were in before the arrow was shot.

There is **no** need to write any code that checks if the monster has been shot when the player chooses to shoot either to the east, to the west or to the south.

Task 7: test 1
Test that the changes you have made work by conducting the following test:

- play the training game
- shoot an arrow
- choose a direction of N for the arrow.

Task 8: test 2
Test that the changes you have made work by conducting the following test:

- play the training game
- move east
- shoot an arrow
- choose a direction of N for the arrow
- shoot an arrow.

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `DisplayMoveOptions`.

(1)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(2)

(iii) Your amended PROGRAM SOURCE CODE for the class `Character`.

(8)

(iv) Your amended PROGRAM SOURCE CODE for the subroutine `Play`.

(6)

(v) SCREEN CAPTURE(S) showing the results of **Test 1**.

(1)

(vi) SCREEN CAPTURE(S) showing the results of **Test 2**.

(1)
(Total 35 marks)

M1.

(a) (i) **Marks are for AO3 (programming)**

1 mark: Selection structure with one correct condition;
1 mark: Both conditions correct and correct logical operator(s);
1 mark: Subroutine returns the correct `True / False` value under all conditions;

A New conditions added to existing selection structure

VB.Net
```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M") Then
    ValidMove = False
  End If
  If Direction = "W" And
Player.GetPosition.NoOfCellsEast = 0 Then
    ValidMove = False
  End If
  Return ValidMove
End Function
```
3

(ii) **Marks are for AO3 (programming)**

1 mark: Selection structure with correct condition added in correct place in the code;
1 mark: Correct error message displayed which will be displayed when move is invalid, and only when the move is invalid;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net
```
  ...
  ValidMove = CheckValidMove(MoveDirection)
  If Not ValidMove Then
    Console.WriteLine("That is not a valid move, please try
again")
  End If
Loop Until ValidMove
...
```
2

(iii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (a)(i) and (a)(ii), including

prompts on screen capture matching those in code. Code for (a)(i) and (a)(ii) must be sensible.

Screen capture(s) showing the error message being displayed after the player tried to move to the west from a cell at the western end of the cavern;

A Alternative output messages if match code for (a)(ii)

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: `SleepyEnemy` class created;
1 mark: Inheritance from `Enemy` class;
1 mark: `MovesTillSleep` property declared;
1 mark: Subroutine `MakeMove` that overrides the one in the base class;
1 mark: `MovesTillSleep` decremented in the `MakeMove` subroutine;
1 mark: Selection structure in `MakeMove` that calls `ChangeSleepStatus` if the value of `MovesTillSleep` is 0; **A** Changing `Awake` property instead of call to `ChangeSleepStatus`
1 mark: Subroutine `ChangeSleepStatus` that overrides the one in the base class;
1 mark: Value of `MovesTillSleep` set to `4` in the `ChangeSleepStatus` subroutine;

I Case of identifiers
A Minor typos in identifiers

VB.Net
```
Class SleepyEnemy
  Inherits Enemy
  Private MovesTillSleep As Integer

  Public Overrides Sub MakeMove(ByVal PlayerPosition As
CellReference)
    MyBase.MakeMove(PlayerPosition)
    MovesTillSleep = MovesTillSleep - 1
    If MovesTillSleep = 0 Then
      ChangeSleepStatus()
    End If
  End Sub

  Public Overrides Sub ChangeSleepStatus()
    MyBase.ChangeSleepStatus()
    MovesTillSleep = 4
  End Sub
End Class
```

8

(ii) **Marks are for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (b)(i), including prompts on screen capture matching those in code. Code for (b)(i) must be sensible.

1 mark: Screen capture(s) showing the player moving east and then east again at the start of the training game. The monster then wakes up and moves two cells nearer to the player. The player then moves south;

1 mark: The monster moves two cells nearer to the player and then disappears from the cavern display;

2

(c) (i) **Mark is for AO3 (programming)**

Appropriate option added to menu;

VB.Net

```
Public Sub DisplayMoveOptions()
  Console.WriteLine()
  Console.WriteLine("Enter N to move NORTH")
  Console.WriteLine("Enter S to move SOUTH")
  Console.WriteLine("Enter E to move EAST")
  Console.WriteLine("Enter W to move WEST")
  Console.WriteLine("Enter A to shoot an arrow")
  Console.WriteLine("Enter M to return to the Main Menu")
  Console.WriteLine()
End Sub
```

1

(ii) **Marks are for AO3 (programming)**

1 mark: Direction of A is allowed;
1 mark: Direction of A allowed only if player has got an arrow;

Maximum 1 mark: If any other invalid moves would be allowed or any valid moves not allowed

VB.Net

```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M" Or Direction = "A") Then
    ValidMove = False
  End If
  If Direction = "A" And Not Player.GetHasArrow Then
    ValidMove = False
  End If
  Return ValidMove
End Function
```

2

(iii) **Marks are for AO3 (programming)**

1 mark: Property `HasArrow` created;
1 mark: `HasArrow` set to `True` when an object is instantiated;

1 mark: Subroutine `GetHasArrow` created;
1 mark: `GetHasArrow` returns the value of `HasArrow`;
1 mark: Subroutine `GetArrowDirection` created;
1 mark: `GetArrowDirection` has an appropriate output message and then gets a value entered by the user;
1 mark: In `GetArrowDirection`, value keeps being obtained from user until it is one of N, S, W or E;
1 mark: `HasArrow` is set to `False` in `GetArrowDirection`;

I Additional output messages
I Case of identifiers
A Minor typos in identifiers

VB.Net
```
Class Character
  Inherits Item
  Private HasArrow As Boolean
  Public Sub MakeMove(ByVal Direction As Char)
    Select Case Direction
    Case "N"
      NoOfCellsSouth = NoOfCellsSouth - 1
    Case "S"
      NoOfCellsSouth = NoOfCellsSouth + 1
    Case "W"
      NoOfCellsEast = NoOfCellsEast - 1
    Case "E"
      NoOfCellsEast = NoOfCellsEast + 1
    End Select
  End Sub

  Public Sub New()
    HasArrow = True
  End Sub

  Public Function GetHasArrow() As Boolean
    Return HasArrow
  End Function

  Public Function GetArrowDirection() As Char
    Dim Direction As Char
    Do
      Console.Write("What direction (E, W, S, N) would you like
to shoot in?")
      Direction = Console.ReadLine
    Loop Until Direction = "E" Or Direction = "W" Or Direction
= "S" Or Direction = "N"
    HasArrow = False
    Return Direction
  End Function
End Class
```
8

(iv) **Marks are for AO3 (programming)**

1 mark: Check for `A` having been entered – added in a sensible place in the code;
1 mark: If `A` was entered there is a call to `GetArrowDirection`;
1 mark: Selection structure that checks if the arrow direction is `N`;

1 mark: Detects if the monster is in any of the cells directly north of the player's current position;
1 mark: If the monster has been hit by an arrow then the correct output message is displayed and the value of `FlaskFound` is set to `True`;
1 mark: The code for moving the player and updating the cavern display is inside an *else* structure (or equivalent) so that this code is not executed if the player chooses to shoot an arrow;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net
```
If MoveDirection  "M" Then
  If MoveDirection = "A" Then
    MoveDirection = Player.GetArrowDirection
    Select MoveDirection
      Case "N"
        If Monster.GetPosition.NoOfCellsSouth
        Console.WriteLine("You have shot the monster and it
cannot stop you finding the flask")
        FlaskFound = True
      End If
    End Select
  Else
    Cavern.PlaceItem(Player.GetPosition, " ")
    Player.MakeMove(MoveDirection)
    Cavern.PlaceItem(Player.GetPosition, "*")
    Cavern.Display(Monster.GetAwake)
    FlaskFound = Player.CheckIfSameCell(Flask.GetPosition)
  End If
  If FlaskFound Then
  ...
```
6

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing the user shooting an arrow northwards at the start of the training game and the message about the monster being shot is displayed;

A Alternative output messages if match code for (c)(iv)

1

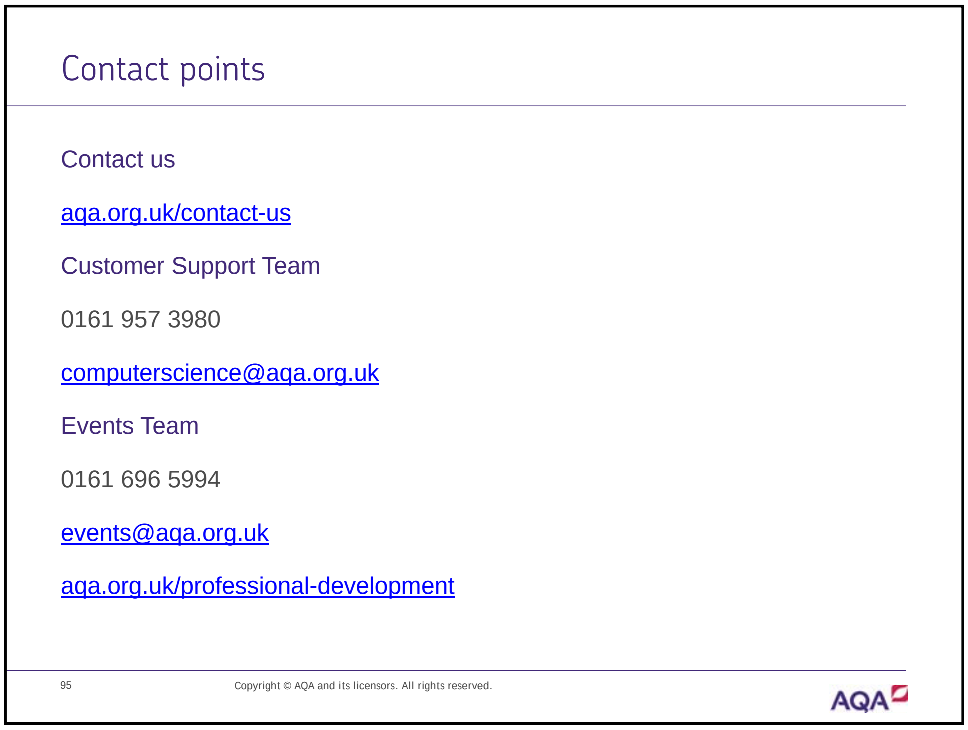
(vi) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing an arrow being shot, no message about the monster being hit is displayed and then the invalid move message is displayed when the player tries to shoot an arrow for a second time;

1

[35]



Contact points

Contact us

aqa.org.uk/contact-us

Customer Support Team

0161 957 3980

computerscience@aqa.org.uk

Events Team

0161 696 5994

events@aqa.org.uk

aqa.org.uk/professional-development

95