Version 1.0 0712

General Certificate of Education (A-level)
June 2012

Computing COMP1

(Specification 2510)

Unit 1: Problem Solving, Programming, Data Representation and Practical Exercise

Final

Mark Scheme

Mark schemes are prepared by the Principal Examiner and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all examiners participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the candidates' responses to questions and that every examiner understands and applies it in the same correct way. As preparation for standardisation each examiner analyses a number of candidates' scripts: alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, examiners encounter unusual answers which have not been raised they are required to refer these to the Principal Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of candidates' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

Further copies of this Mark Scheme are available from: aqa.org.uk

Set and published by the Assessment and Qualifications Alliance.

The Assessment and Qualifications Alliance (AQA) is a company limited by guarantee registered in England and Wales (company number 3644723) and a registered charity (registered charity number 1073334).
Registered address: AQA, Devas Street, Manchester M15 6EX.

To Examiners:

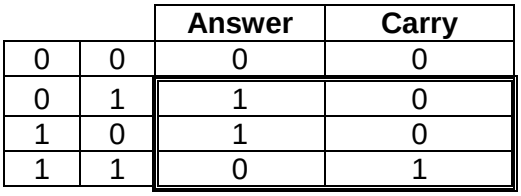
1. **When to award '0' (zero) when inputting marks on QMS *and on scripts*:** A mark of 0 should be awarded where a candidate has attempted a question but failed to write anything creditworthy. Insert a hyphen when a candidate has not attempted a question. By these two actions the Principal Examiner will be able to distinguish between the two (nothing creditworthy/unattempted) when analysing any statistics.

2. This mark scheme contains the correct responses which we believe that candidates are most likely to give. Other valid responses are possible to some questions and should be credited. Examiners should refer off-mark scheme responses that they believe are creditworthy to a Team Leader.

The following annotation may be used in the mark scheme:

; - means a single mark
// - means alternative response
/ - means an alternative word or sub-phrase
A - means acceptable creditworthy answer
R - means reject answer as not creditworthy
NE - means not enough
I - means ignore
DPT -means 'Don't penalise twice'

No marks will be awarded for answers to testing questions where there is no evidence of programming code for the question(s) asked or where the screen captures provided by the candidate do not match what would be produced by the programming code.

Qu	Part	Marking Guidance	Marks
1	01		**3**

		Answer	Carry	
0	0	0	0	
0	1	1	0	;
1	0	1	0	;
1	1	0	1	;

A. 10 instead of 0 in the Answer column for the final row of the table

2	02	011 0010; **R.** If not 7 bits	**1**
	03	1011 0000 **Mark as follows:** Correct data bits; Correct parity bit for the candidate's data bits; **R.** If not 8 bits	**2**
	04	Error correction (not just error detection) (for single errors); Can detect when two errors have occurred in data transmission; Reduces the need for the retransmission of data; Decreases the likelihood of an undetected error // improved error	

		detection; Can locate an error (not just detect that an error has occurred);	**MAX 1**

3	05	300; * 2; // 600;; NOTE: award 1 mark for doubling an incorrectly calculated highest frequency	**2**
	06	Regular samples are taken (of the analogue signal); Samples are quantised // the height of each sample is approximated to an integer value // height of samples measured // amplitude/volume measured; Each integer value is encoded as a binary value // measurements are coded in a fixed number of bits; output the binary numbers as digital signals/voltage levels;	**MAX 3**
	07	Can (easily) synthesise musical notation from it; Can be played on different instruments; Can be (easily) transposed to a different key/pitch; Produces (relatively) small files; Easy to manipulate (the data); Allows for easy interface with electronic musical instruments; No data lost about a musical note;	**MAX 1**
	08	Length/duration (of note) // Note-on and Note-off; Instrument; Velocity//Speed; Volume//Amplitude; Timbre; Pedal effects; Channel; Instructions about how to recreate a sound; Aftertouch; Pitch bend; Note envelope; **R.** Note/key/pitch/frequency; **A.** Other sensible answers;	**MAX 1**

4	09	<table><tr><th>Original State</th><th>Input</th><th>New State</th></tr><tr><td>S0</td><td>10</td><td>S10</td></tr><tr><td>S0</td><td>20</td><td>S20</td></tr><tr><td>S0</td><td>50</td><td>S50</td></tr><tr><td>S0</td><td>R</td><td>S0</td></tr></table> ; ; ; *Note: order of completed rows not important*	**3**

	10	20, 20, 10; R, R, 50;	

		10, 20, 20; 20, 50, 50; 20, R, 50;	**MAX 4**

5	11	(Each pixel) can be one of $4/2^2$ possible colours/values // Two bits are needed to represent the 4 possible bit patterns/colours/values // because there are 4/more than 2 colours in the image;	**1**
	12	1 1 1 1 1 1 0 0 0 0 1 1 **1** **0** 1 1 ;; // 1 1 1 1 1 1 0 0 0 0 1 1 **0** **1** 1 1 ;; **Mark as follows:** 13th and 14th bits correct; Other bits correct;	**2**
	13	8*8 =64; * 2 = 128; ÷ 8 = 16; // 8*8*2÷ 8;;; 16;;; **A.** 128 bits as being worth 2 marks	**3**
	14	(Type of) shape // rectangle // square; Coordinates of corner/corners // position of a corner // top left coordinates; Identifier; Length of side(s) // width // height // coordinates of an opposing corner; Line colour // outer colour; Line width; Fill colour // inner colour; Angle of rotation; **A.** coordinates of midpoint/centre; **A.** radius/diameter **A.** circle/oval **NE.** Position/coordinates **NE.** Colour	**MAX 3**
	15	(For geometric images) less storage space/memory likely to be needed; **NE.** less space (For geometric images) will load faster from secondary storage; (For geometric images) will download faster; Can be scaled/resized without distortion; **A.** zoom Image can be (more easily) searched for particular objects; Can (more easily) manipulate individual objects in an image;	**MAX 2**

6	16	Correct variable declarations for `Bit`, `Answer` and `Column`; **I.** additional variable declarations `Column` initialised correctly before the start of the loop; `Answer` initialised correctly before the start of the loop;	

		`While`/`Repeat` loop, with syntax allowed by the programming language used, after the variable initialisations; and correct condition for the termination of the loop; **R.** `For` loop **A.** any `While`/`Repeat` loop with logic corresponding to that in flowchart (for a loop with a condition at the start accept >=1 or >0 but reject <>0) Correct prompt "`Enter bit value: `"; followed by `Bit` assigned value entered by user; followed by `Answer` given new value; **R.** if incorrect value would be calculated followed by value of `Column` divided by 2; **A.** multiplying by 0.5 correct prompt and the assignment statements altering `Bit`, `Answer` and `Column` are all within the loop; After the loop - output message followed by value of `Answer`; **I.** Case of variable names, player names and output messages **A.** Minor typos in variable names and output messages **I.** spacing in prompts **A.** answers where formatting of decimal values is included e.g. `Writeln('Decimal value is: ', Answer : 3)` **A.** initialisation of variables at declaration stage **A.** no brackets around `column * bit`	**11**
	17	****SCREEN CAPTURE**** *Must match code from 16, including prompts on screen capture matching those in code* **Mark as follows:** 'Enter bit value: ' + first user input of 1 'Enter bit value: ' + second user input of 1; 'Enter bit value: ' + third user input of 0 'Enter bit value: ' + fourth user input of 1; Value of 13 outputted;	**3**
	18	15;	**1**
	19	16 // twice as many // double;	**1**
	20	Design; **A.** Planning	**1**
	21	Implementation;	**1**

7	22	`ResetCavern;` (all languages) `// GetNewRandomPosition` (Pascal only) `// WriteWithMsg` (VB6 only) `// WriteLineWithMsg` (VB6 only) `// WriteLine` (VB6 only) `// WriteNoLine` (VB6 only) `// ReadLine` (VB6 only); `// SetUpTrapPostions` (Python / Java only); **R.** if any additional code (including routine interface) **R.** if spelt incorrectly	

		I. case	**1**
	23	`DisplayMenu` // `DisplayMoveOptions` // `DisplayWonGameMessage` // `DisplayTrapMessage` // `DisplayLostGameMessage` // `WriteWithMsg` (VB6 only) // `WriteLineWithMsg` (VB6 only) // `WriteLine` (VB6 only) // `WriteNoLine` (VB6 only); **A.** `DisplayCavern;` **R.** if any additional code (including routine interface) **R.** if spelt incorrectly **I.** case	**1**
	24	`Count1` // `Count2` // `Count`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	**1**
	25	`Cavern` // `TrapPositions`; **R.** if any additional code (including routine interface) **R.** if spelt incorrectly **A.** `LoadedGameData.TrapPositions` **A.** `CurrentGameData.TrapPositions` **I.** case	**1**
	26	When the value of the cell in the `Cavern` array // When the value of the cell to place the item in; Indicated by the `Position` variable; Contains a space // does not contain another item; **R.** is empty **MAX 2** if <u>no</u> variable names used in description	**3**
	27	The number of times to repeat is <u>known</u>; **A.** fixed	**1**
	28	Makes the program code easier to understand; Makes it easier to update the program; Makes it easier to change the number of traps (in the game);	**MAX 1**
	29	In text files all data is stored as strings/ASCII values/lines/characters // Text files use only byte values that display sensibly on a VDU // stores only values that can be opened and read in a text editor; Binary files store data using different data types; **A.** by example **A.** Binary files can only be correctly interpreted by application that created them	**2**
	30	Easier reuse of routines in other programs; Routine can be included in a library; Helps to make the program code more understandable; Ensures that the routine is self-contained // routine is independent of the rest of the program;	

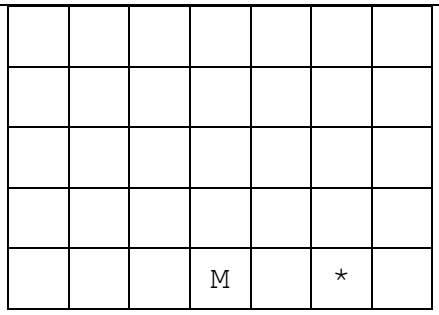
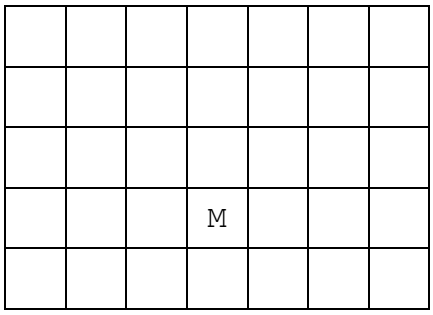
		(Global variables use memory while a program is running) but local variables use memory for only part of the time a program is running; reduces possibility of undesirable side effects; Using global variables makes a program harder to debug;	**MAX 2**
	31	(If it was not then) `MonsterAwake` is set to the Boolean value returned by the second call to `CheckIfSameCell`; this would overwrite any True value returned by the first call to `CheckIfSameCell`; // Otherwise the monster would never wake up when the player triggers the first trap;; // Otherwise the monster would only wake up when the player triggers the second trap;;	**2**

8	32	Appropriate option added; **A.** Any sensible prompt **A.** Prompt added anywhere in subroutine **R.** If prompt asks for character other than D	**1**
	33	Additional case statement for option D added correctly and all of the rest of the code added inside the correct option of the case statement; **A.** any character instead of D (except N, S, W, E) – only if matches prompt from 32 `NoOfCellsSouth` incremented within the correct option of the case statement; `NoOfCellsEast` incremented within the correct option of the case statement;	**3**
	34	Additional condition added to `IF` statement ; **A.** answers using an additional `IF` statement **R.** if value of 'D' will result in `False` being returned by function **R.** if function will always return `True`	**1**
	35	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 32, 33 and 34* Screen capture(s) showing modified menu shown to user and option 'D' selected; Screen capture(s) showing both original position of player in the cavern and the new position of the player in the cavern;	**2**

9	36	Selection structure with correct condition; Inside the selection structure there is code that will display the correct message on the screen; **I.** Capitalisation and minor typos in message **R.** different message Selection structure is in the correct place in the code;	**3**
	37	`If` statement with a correct condition;	

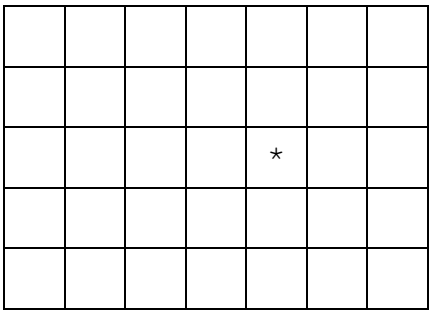
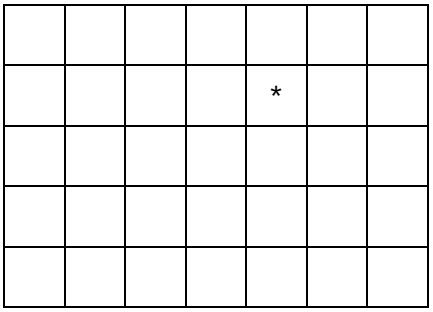
		Correct logic and 2nd condition for `If` statement; Value of `False` returned correctly by the function if illegal north move is made; **R.** if a value of `False` will always be returned by the function **R.** if all north moves will return false **R.** if all moves when `PlayerPosition.NoOfCellsSouth` is in row 1 will return false Value of `True` returned correctly by the function if legal north move is made; **A.** Answers which combine all the checks for a valid move into one `If` statement **I.** missing option '`D`' in code	**4**
	38	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 36 and correct code for 37* Screen capture(s) showing correct cavern state with a player at the northern end of the cavern (top line), 'N' being entered at prompt, followed by correct error message being displayed;	**1**

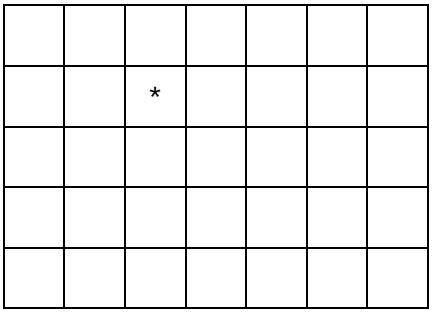
10	39	`NoOfMoves` is assigned the value 0 – before the first repetition structure in `PlayGame`; **I.** Case of variable names **A.** Minor typos in variable name **A.** assignment statement(s) in other subroutine(s) if correct functionality would be obtained `NoOfMoves` incremented in any sensible place in the code inside the first selection structure in `PlayGame` subroutine; One correct message displayed with `NoOfMoves`; Second correct message displayed with `NoOfMoves`; Correct logic – both of the messages will be displayed only under the correct circumstances; **A.** minor typos in messages **I.** capitalisation & spacing in messages **A.** message displayed on more than one line **A.** more than one line of code used to display a message **A.** `NoOfMoves` declared as global **I.** `NoOfMoves` declaration not shown in PROGRAM SOURCE CODE	**5**
	40	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 39* Screen capture(s) showing correct cavern state:	

		 followed by message "`The number of moves you took to find the flask was 3`"; **A.** Different message – if it matches code in 39 and displays final value of `NoOfMoves` correctly **R.** If message "`The number of moves that you survived …`" is also shown	**1**
	41	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 39* Screen capture(s) showing correct cavern state:  followed by message "`The number of moves that you survived in the cavern for was 2`"; **A.** Different message – if it matches code in 39 and displays final value of `NoOfMoves` correctly **R.** If message "`The number of moves you took…`" is also shown	**1**

11	42	`CalculateDistance` subroutine created – with begin and end of subroutine; `PlayerPosition` and `MonsterPosition` passed as parameters to the `CalculateDistance` subroutine; **I.** additional unnecessary parameters **R.** global variables **A.** four integer values instead of two `CellReference` values **R.** passing by value for parameters of type `CellReference` (VB6 only) Integer value returned by subroutine either as parameter passed by reference or by function return value; **R.** global variable **A.** real value	

		Calculates difference between the `NoOfCellsEast` for the monster and the player; **R.** if the result can be a negative distance Calculates difference between the `NoOfCellsSouth` for the monster and the player; **R.** if the result can be a negative distance Calculates the total distance between the monster and the player; **A.** Incorrect values for differences in `NoOfCellsEast` and `NoOfCellsSouth` being added together Distance calculated is actually returned by the subroutine; **A.** use of global variable **I.** Case of identifiers **A.** Minor typos in identifiers **I.** Order of parameters in routine interface	**7**
	43	Call to `CalculateDistance` subroutine; **R.** if parameter list does not match answer to 42 Displays "`Distance between monster and player: `" in correct place; **A.** any place in code after call to `DisplayMoveOptions` and before call to `MakeMove` **A.** minor typos in prompt **I.** capitalisation Displays the calculated distance; **R.** if no evidence of any calculation for the distance **R.** if distance is displayed before call to `CalculateDistance` subroutine **R.** if distance returned by `CalculateDistance` stored in a global variable **R.** if distance calculated in part 42 would not actually be displayed e.g. program would not compile/run **A.** use of temporary variable to store the value returned by `CalculateDistance` with contents of temporary variable then displayed using output message **I.** Case of identifiers and output messages **A.** Minor typos in output messages **I.** spacing in output messages	**3**

	44	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 42 and/or 43* Player shown in the cell 3 south and 5 east of the northwest corner AND "Distance between monster and player: 3" shown;  **I.** monster symbol (M) displayed in the cavern	**1**
	45	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 42 and/or 43* Player shown in the cell 2 south and 5 east of the northwest corner AND "Distance between monster and player: 2" shown;  **I.** monster symbol (M) displayed in the cavern	**1**

	46	****SCREEN CAPTURE(S)**** *This is conditional on sensible code for 42 and/or 43* Player shown in the cell 2 south and 3 east of the northwest corner AND "Distance between monster and player: 2" shown; **I.** monster displayed in the  symbol (M) cavern	**1**

PASCAL Mark Scheme

Qu	Part	Marking Guidance	Marks

6	16	`Program Question6;`
		`  Var`
		`  Answer : Integer;`
		`  Column : Integer;`
		`  Bit : Integer;`
		`  Begin`
		`  Answer := 0;`
		`  Column := 8;`
		`  Repeat`
		`  Writeln('Enter bit value: ');`
		`  Readln(Bit);`
		`  Answer := Answer + (Column * Bit);`
		`  Column := Column DIV 2;`
		`  Until Column < 1;`
		`  Writeln('Decimal value is: ', Answer);`
		`  Readln;`
		`  End.`
			11

8	32	`Procedure DisplayMoveOptions;`
		`  Begin`
		`  Writeln;`
		`  Writeln('Enter N to move NORTH');`
		`  Writeln('Enter E to move EAST');`
		`  Writeln('Enter S to move SOUTH');`
		`  Writeln('Enter W to move WEST');`
		`  Writeln('Enter D to move SOUTHEAST');`
		`  Writeln('Enter M to return to the Main Menu');`
		`  Writeln;`
		`  End;`
			1

	33	`Case Direction Of`
		`  'N' : PlayerPosition.NoOfCellsSouth :=`
		`PlayerPosition.NoOfCellsSouth - 1;`
		`  'S' : PlayerPosition.NoOfCellsSouth :=`
		`PlayerPosition.NoOfCellsSouth + 1;`
		`  'W' : PlayerPosition.NoOfCellsEast :=`
		`PlayerPosition.NoOfCellsEast - 1;`
		`  'E' : PlayerPosition.NoOfCellsEast :=`
		`PlayerPosition.NoOfCellsEast + 1;`
		`  'D' : Begin`
		`  PlayerPosition.NoOfCellsSouth :=`
		`PlayerPosition.NoOfCellsSouth + 1;`
		`  PlayerPosition.NoOfCellsEast :=`
		`PlayerPosition.NoOfCellsEast + 1;`
		`  End;`
		`End;`
			3

	34	ValidMove := True; If Not (Direction In ['N','S','W','E',**'D'**,'M']) Then ValidMove := False; CheckValidMove := ValidMove;	**1**

9	36	Repeat
		  DisplayMoveOptions;
		  MoveDirection := GetMove;
		  ValidMove := CheckValidMove(PlayerPosition, MoveDirection);
		  If Not ValidMove
		  Then Writeln('That is not a valid move, please try again');
		Until ValidMove;
		
		Alternative answer
		If ValidMove = False...
			3

	37	ValidMove := True;
		If Not (Direction In ['N','S','W','E','D','M'])
		  Then ValidMove := False;
		If (PlayerPosition.NoOfCellsSouth = 1) And (Direction = 'N')
		  Then ValidMove := False;
		CheckValidMove := ValidMove;
		
		Alternative answer
		If ValidMove And (Direction = 'N')
		  Then ValidMove := ValidMove And
		  (PlayerPosition <> 1);
			4

10	39	Eaten:= False;
		FlaskFound := False;
		DisplayCavern(Cavern, MonsterAwake);
		NoOfMoves := 0;
		Repeat
		  ...
		  If MoveDirection <> 'M'
		  Then
		  Begin
		  MakeMove(Cavern, MoveDirection, PlayerPosition);
		  NoOfMoves := NoOfMoves + 1;
		  DisplayCavern(Cavern, MonsterAwake);
		  ...
		  If FlaskFound
		  Then
		  Begin
		  DisplayWonGameMessage;	

```
              Writeln('The number of moves you took
to find the flask was ',NoOfMoves);
            End;
            ...
            If Eaten
              Then
                Begin
                  DisplayLostGameMessage;
                  Writeln('The number of moves you
survived in the cavern for was ', NoOfMoves);
                End;
```

Alternative answer

```
Until Eaten Or FlaskFound Or (MoveDirection = 'M');
If Eaten
  Then Writeln('The number of moves that you
survived in the cavern for was ', NoOfMoves);
If FlaskFound
  Then Writeln('The number of moves you took to
find the flask was ', NoOfMoves);
```

Alternative answer

```
  If FlaskFound
    Then DisplayWonGameMessage(NoOfMoves);
  ...
  If Eaten
    Then DisplayLostGameMessage(NoOfMoves);
```

together with modified `DisplayWonGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

```
Procedure DisplayWonGameMessage(NoOfMoves :
Integer);
  Begin
    Writeln('Well done!  You have found the flask
containing the Styxian potion.');
    Writeln('You have won the game of MONSTER!');
    Writeln('The number of moves you took to find
the flask was ',NoOfMoves);
    Writeln;
  End
```

and modified `DisplayLostGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

```
Procedure DisplayLostGameMessage(NoOfMoves :
Integer);
  Begin
    Writeln('ARGHHHHHH!  The monster has eaten you.
GAME OVER.');
```

		Writeln('Maybe you will have better luck next time you play MONSTER!'); **Writeln('The number of moves you survived in the cavern for was ', NoOfMoves);** Writeln; End;	**5**

11	42	Function CalculateDistance(PlayerPosition, MonsterPosition : TCellReference) : Integer;
		Var
		Distance : Integer;
		Begin
		If PlayerPosition.NoOfCellsEast > MonsterPosition.NoOfCellsEast
		Then Distance := PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast
		Else Distance := MonsterPosition.NoOfCellsEast - PlayerPosition.NoOfCellsEast;
		If PlayerPosition.NoOfCellsSouth > MonsterPosition.NoOfCellsSouth
		Then Distance := Distance + PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth
		Else Distance := Distance + MonsterPosition.NoOfCellsSouth - PlayerPosition.NoOfCellsSouth;
		CalculateDistance := Distance;
		End;
		
		Alternative answer
		Distance := Abs(PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast) + Abs(PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth));
		
		Alternative answer
		Distance := Trunc(Sqrt(Sqr(PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast)) + Sqrt(Sqr(PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth)));
		
		Alternative answer
		Distance := Round(Sqrt(Sqr(PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast)) + Sqrt(Sqr(PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth)));
		
		Alternative answer	

		Distance2 : Integer;
		...
		Distance := PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast;
		If Distance < 0
		  Then
		  Distance := Distance * -1;
		Distance2 := PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth;
		If Distance2 < 0
		  Then
		  Distance2 := Distance2 * -1;
		Distance := Distance + Distance2;
			7

	43	DisplayMoveOptions; **Writeln('Distance between monster and player: ', CalculateDistance(PlayerPosition, MonsterPosition));**	**3**

VB.NET Mark Scheme

Qu	Part	Marking Guidance	Marks

6	16	Sub Main()
		  Dim Answer As Integer
		  Dim Column As Integer
		  Dim Bit As Integer
		  Answer = 0
		  Column = 8
		  Do
		  Console.Write("Enter bit value: ")
		  Bit = Console.ReadLine
		  Answer = Answer + (Column * Bit)
		  Column = Column / 2
		  Loop Until Column < 1
		  Console.Write("Decimal value is: " & Answer)
		  Console.ReadLine()
		End Sub
		
		Alternative Answer
		Column = Column \ 2
			11

8	32	Sub DisplayMoveOptions()
		  Console.WriteLine()
		  Console.WriteLine("Enter N to move NORTH")
		  Console.WriteLine("Enter E to move EAST")
		  Console.WriteLine("Enter S to move SOUTH")
		  Console.WriteLine("Enter W to move WEST")
		  Console.WriteLine("Enter D to move SOUTHEAST")
		  Console.WriteLine("Enter M to return to the Main Menu")
		  Console.WriteLine()
		End Sub
			1

	33	Case "E"
		  PlayerPosition.NoOfCellsEast = PlayerPosition.NoOfCellsEast + 1
		Case "D"
		  PlayerPosition.NoOfCellsSouth = PlayerPosition.NoOfCellsSouth + 1
		  PlayerPosition.NoOfCellsEast = PlayerPosition.NoOfCellsEast + 1
			3

	34	ValidMove = True
		If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M" **Or Direction = "D")** Then
		  ValidMove = False
		End If
		CheckValidMove = ValidMove
			1

9	36	Do
		  DisplayMoveOptions()
		  MoveDirection = GetMove()
		  ValidMove = CheckValidMove(PlayerPosition, MoveDirection)
		  If Not ValidMove Then
		  Console.WriteLine("That is not a valid move, please try again")
		  End If
		Loop Until ValidMove
			3

	37	If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "D" Or Direction = "M") Then
		  ValidMove = False
		End If
		If PlayerPosition.NoOfCellsSouth = 1 And Direction = "N" Then
		  ValidMove = False
		End If
		CheckValidMove = ValidMove
		
		Alternative answer
		If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M") Then
		  ValidMove = False
		End If
		If ValidMove And (Direction = "N") Then
		  ValidMove = (ValidMove And (PlayerPosition.NoOfCellsSouth <> 1))
		End If
			4

10	39	Dim ValidMove As Boolean
		Eaten = False
		FlaskFound = False
		DisplayCavern(Cavern, MonsterAwake)
		NoOfMoves = 0
		Do
		...
		If MoveDirection <> "M" Then
		  MakeMove(Cavern, MoveDirection, PlayerPosition)
		  NoOfMoves = NoOfMoves + 1
		  DisplayCavern(Cavern, MonsterAwake)
		...
		If FlaskFound Then
		  DisplayWonGameMessage()
		  Console.WriteLine("The number of moves you took to find the flask was " & NoOfMoves)
		End If	

```
...

If Eaten Then
  DisplayLostGameMessage()
  Console.WriteLine("The number of moves that you
survived in the cavern for was " & NoOfMoves)
End If
...
```

Alternative answer

```
Loop Until Eaten Or FlaskFound Or MoveDirection =
"M"
If Eaten Then
  Console.WriteLine("The number of moves that you
survived in the cavern for was " & NoOfMoves)
End If
If FlaskFound Then
  Console.WriteLine("The number of moves you took
to find the flask was " & NoOfMoves)
End If
```

Alternative answer

```
If FlaskFound Then
  DisplayWonGameMessage(NoOfMoves)
End If
...
If Eaten Then
  DisplayLostGameMessage(NoOfMoves)
End If
```

together with modified `DisplayWonGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayWonGameMessage(ByVal NoOfMoves As
Integer)
  Console.WriteLine("Well done!  You have found the
flask containing the Styxian potion.")
  Console.WriteLine("You have won the game of
MONSTER!")
  Console.Writeline("The number of moves you took
to find the flask was " & NoOfMoves)
  Console.WriteLine()
End Sub
```

and modified `DisplayLostGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayLostGameMessage(ByVal NoOfMoves As
Integer)
  Console.WriteLine("ARGHHHHHH!  The monster has
```

```
eaten you.  GAME OVER.")
  Console.WriteLine("Maybe you will have better
luck next time you play MONSTER!")
  Console.WriteLine("The number of moves you
survived in the cavern for was " & NoOfMoves);
  Console.WriteLine()
End Sub
```

5

11 42

```
Function CalculateDistance(ByVal PlayerPosition As
CellReference, ByVal MonsterPosition As
CellReference) As Integer

  Dim Distance As Integer
  If PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast Then
    Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
  Else
    Distance = MonsterPosition.NoOfCellsEast -
PlayerPosition.NoOfCellsEast
  End If
  If PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth Then
    Distance = Distance +
PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
  Else
    Distance = Distance +
MonsterPosition.NoOfCellsSouth -
PlayerPosition.NoOfCellsSouth
  End If
  CalculateDistance = Distance
End Function
```

Alternative answer

```
Distance =
System.Math.Abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
System.Math.Abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

A. this alternative answer if `System.Math` included
A. give benefit of doubt for this answer if no evidence of `System.Math` included

Alternative answer

```
Distance = (((PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) ^ 2) ^ 0.5) +
(((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth) ^ 2) ^ 0.5)
```

		Alternative answer
		`Dim Distance2 As Integer`
		`...`
		`Distance = PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast`
		`If Distance < 0 Then`
		`  Distance = Distance * -1`
		`End If`
		`Distance2 = PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth`
		`If Distance2 < 0 Then`
		`  Distance2 = Distance2 * -1`
		`End If`
		`Distance = Distance + Distance2`
			7

	43	`DisplayMoveOptions()` **`Console.WriteLine("Distance between monster and player: " & CalculateDistance(PlayerPosition, MonsterPosition))`**	**3**

VB6 Mark Scheme

Qu	Part	Marking Guidance	Marks

6	16	Private Sub Form_Load()
		  Dim Answer As Integer
		  Dim Column As Integer
		  Dim Bit As Integer
		  Answer = 0
		  Column = 8
		  Do
		  Bit = InputBox("Enter bit value: ")
		  Answer = Answer + (Column * Bit)
		  Column = Column / 2
		  Loop Until Column < 1
		  MsgBox ("Decimal value is: " & Answer)
		End Sub
		
		Alternative Answer
		Column = Column \ 2
			11

8	32	Private Sub DisplayMoveOptions()
		  WriteLine ("")
		  WriteLine ("Enter N to move NORTH")
		  WriteLine ("Enter E to move EAST")
		  WriteLine ("Enter S to move SOUTH")
		  WriteLine ("Enter W to move WEST")
		  WriteLine ("Enter D to move SOUTHEAST")
		  WriteLine ("Enter M to return to the Main Menu")
		  WriteLine ("")
		End Sub
		
		A. Text1.Text = Text1.Text & "Enter D to move SOUTHEAST"
			1

	33	Case "E"
		  PlayerPosition.NoOfCellsEast = PlayerPosition.NoOfCellsEast + 1
		Case "D"
		  PlayerPosition.NoOfCellsSouth = PlayerPosition.NoOfCellsSouth + 1
		  PlayerPosition.NoOfCellsEast = PlayerPosition.NoOfCellsEast + 1
			3

	34	ValidMove = True
		If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M" **Or Direction = "D")** Then
		  ValidMove = False
		End If
		CheckValidMove = ValidMove
			1

9	36	Do
		  Call DisplayMoveOptions()
		  MoveDirection = GetMove()
		  ValidMove = CheckValidMove(PlayerPosition, MoveDirection)
		  If Not ValidMove Then
		  WriteLine("That is not a valid move, please try again")
		  End If
		Loop Until ValidMove
		
		A. Text1.Text = Text1.Text & "That is not a valid move, please try again"
		A. WriteLineWithMsg
			3

	37	If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "D" Or Direction = "M") Then
		  ValidMove = False
		End If
		If PlayerPosition.NoOfCellsSouth = 1 And Direction = "N" Then
		  ValidMove = False
		End If
		CheckValidMove = ValidMove
		
		Alternative answer
		If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M") Then
		  ValidMove = False
		End If
		If ValidMove And (Direction = "N") Then
		  ValidMove = (ValidMove And (PlayerPosition.NoOfCellsSouth <> 1))
		End If
			4

10	39	Dim ValidMove As Boolean
		Eaten = False
		FlaskFound = False
		Call DisplayCavern(Cavern, MonsterAwake)
		NoOfMoves = 0
		Do
		...
		If MoveDirection <> "M" Then
		  Call MakeMove(Cavern, MoveDirection, PlayerPosition)
		  NoOfMoves = NoOfMoves + 1
		  Call DisplayCavern(Cavern, MonsterAwake)
		...	

```
If FlaskFound Then
  Call DisplayWonGameMessage()
  WriteLine("The number of moves you took to find
the flask was " & NoOfMoves)
End If

...

If Eaten Then
  Call DisplayLostGameMessage()
  WriteLine("The number of moves that you survived
in the cavern for was " & NoOfMoves)
End If
...
```

Alternative answer

```
Loop Until Eaten Or FlaskFound Or MoveDirection =
"M"
If Eaten Then
  WriteLine("The number of moves that you survived
in the cavern for was " & NoOfMoves)
End If
If FlaskFound Then
  WriteLine("The number of moves you took to find
the flask was " & NoOfMoves)
End If
```

Alternative answer

```
If FlaskFound Then
  DisplayWonGameMessage(NoOfMoves)
End If
...
If Eaten Then
  DisplayLostGameMessage(NoOfMoves)
End If
```

together with modified `DisplayWonGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayWonGameMessage(ByVal NoOfMoves As
Integer)
  WriteLine("Well done!  You have found the flask
containing the Styxian potion.")
  WriteLine("You have won the game of MONSTER!")
  Writeline("The number of moves you took to find
the flask was " & NoOfMoves);
  WriteLine("")
End Sub
```

and modified `DisplayLostGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

		Sub DisplayLostGameMessage(**ByVal NoOfMoves As Integer**)
		  WriteLine("ARGHHHHHH! The monster has eaten you. GAME OVER.")
		  WriteLine("Maybe you will have better luck next time you play MONSTER!")
		  WriteLine("The number of moves you survived in the cavern for was " & NoOfMoves);
		  WriteLine("")
		End Sub
		
		A. Text1.Text = Text1.Text & "The number of moves that you survived in the cavern for was"
		A. Text1.Text = Text1.Text & "The number of moves you took to find the flask was "
		A. WriteLineWithMsg
			5

11	42	Private Function CalculateDistance(ByRef PlayerPosition As CellReference, ByRef MonsterPosition As CellReference) As Integer
		
		  Dim Distance As Integer
		  If PlayerPosition.NoOfCellsEast > MonsterPosition.NoOfCellsEast Then
		  Distance = PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast
		  Else
		  Distance = MonsterPosition.NoOfCellsEast - PlayerPosition.NoOfCellsEast
		  End If
		  If PlayerPosition.NoOfCellsSouth > MonsterPosition.NoOfCellsSouth Then
		  Distance = Distance + PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth
		  Else
		  Distance = Distance + MonsterPosition.NoOfCellsSouth - PlayerPosition.NoOfCellsSouth
		  End If
		  CalculateDistance = Distance
		End Function
		
		Alternative answer
		Distance = (((PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast) ^ 2) ^ 0.5) + (((PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth) ^ 2) ^ 0.5)	

		Alternative answer
		`Distance = Abs(PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast) + Abs(PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth)`
		
		Alternative answer
		`Dim Distance2 As Integer`
		`...`
		`Distance = PlayerPosition.NoOfCellsEast - MonsterPosition.NoOfCellsEast`
		`If Distance < 0 Then`
		`  Distance = Distance * -1`
		`End If`
		`Distance2 = PlayerPosition.NoOfCellsSouth - MonsterPosition.NoOfCellsSouth`
		`If Distance2 < 0 Then`
		`  Distance2 = Distance2 * -1`
		`End If`
		`Distance = Distance + Distance2`
			7

	43	`DisplayMoveOptions()` **`WriteLine("Distance between monster and player: " & CalculateDistance(PlayerPosition, MonsterPosition))`** **A.** `Text1.Text = Text1.Text & "Distance between monster and player: " & CalculateDistance(PlayerPosition, MonsterPosition)` **A.** `WriteLineWithMsg`	**3**

JAVA Mark Scheme

Qu	Part	Marking Guidance	Marks

6	16	`public class Question6 {`
		`  AQAConsole console=new AQAConsole();`
		`  public Question6(){`
		`  int column;`
		`  int answer;`
		`  int bit;`
		`  answer=0;`
		`  column=8;`
		`  do{`
		`  console.print("Enter bit value: ");`
		`  bit=console.readInteger("");`
		`  answer=answer+(column*bit);`
		`  column=column/2;`
		`  }while(column>=1);`
		`  console.print("Decimal value is: ");`
		`  console.println(answer);`
		`  }`
		`  public static void main(String[] arrays){`
		`  new Question6();`
		`  }`
		`}`
			11

8	32	`void displayMoveOptions() {`
		`  console.println();`
		`  console.println("Enter N to move NORTH");`
		`  console.println("Enter E to move EAST");`
		`  console.println("Enter S to move SOUTH");`
		`  console.println("Enter W to move WEST");`
		`  **console.println("Enter D to move SOUTHEAST");**`
		`  console.println("Enter M to return to the Main Menu");`
		`  console.println();`
		`}`
			1

	33	`switch (direction) {`
		`  case 'N':`
		`  playerPosition.noOfCellsSouth--;`
		`  break;`
		`  case 'S':`
		`  playerPosition.noOfCellsSouth++;`
		`  break;`
		`  case 'W':`
		`  playerPosition.noOfCellsEast--;`
		`  break;`
		`  case 'E':`
		`  playerPosition.noOfCellsEast++;`
		`  break;`	

		case 'D': **playerPosition.noOfCellsSouth++;** **playerPosition.noOfCellsEast++;** **break;** }	**3**

	34	validMove = true;
		if (!(direction == 'N' || direction == 'S' || direction == 'W'|| direction == 'E' **|| direction == 'D'** || direction == 'M')) {
		validMove = false;
		}
		return validMove;
			1

9	36	do {
		displayMoveOptions();
		moveDirection = getMove();
		validMove = checkValidMove(playerPosition, moveDirection);
		if (!validMove) {
		console.println("That is not a valid move, please try again");
		}
		} while (!validMove);
		
		Alternative answer
		if (validMove == false)
			3

	37	validMove = true;
		if (!(direction == 'N' || direction == 'S' || direction == 'W'|| direction == 'E' || direction == 'D' || direction == 'M')) {
		validMove = false;
		}
		if (validMove && direction == 'N') {
		validMove = validMove &&
		(playerPosition.noOfCellsSouth != 1);
		}
		return validMove;
		
		Alternative answer
		if (playerPosition.noOfCellsSouth == 1 && direction == 'N') {
		validMove = false;
		}
			4

10	39	eaten = false;
		flaskFound = false;
		displayCavern(cavern, monsterAwake);
		noOfMoves = 0;	

```
do {
  ...
  if (moveDirection != 'M') {
    makeMove(cavern, moveDirection,
playerPosition);
    noOfMoves++;
    displayCavern(cavern, monsterAwake);
   flaskFound = checkIfSameCell(playerPosition,
flaskPosition);
   if (flaskFound) {
     displayWonGameMessage();
     console.println("The number of moves you took
to find the flask was " + noOfMoves);
   }
   ...

   if (eaten) {
     displayLostGameMessage();
     console.println("The number of moves you
survived in the " + "cavern for was " +
noOfMoves);
   }
```

Alternative answer

```
} while (!(eaten || flaskFound || moveDirection ==
'M'));
if (flaskFound) {
  console.println("The number of moves you took to
find the flask was " + noOfMoves);
}
if (eaten) {
  console.println("The number of moves you
survived in the " + "cavern for was " +
noOfMoves);
}
```

Alternative answer

```
eaten = false;
flaskFound = false;
displayCavern(cavern, monsterAwake);
noOfMoves = 0;
do {
  ...
  if (moveDirection != 'M') {
    makeMove(cavern, moveDirection,
playerPosition);
    noOfMoves++;
    displayCavern(cavern, monsterAwake);
    ...
```

together with modified `displayLostGameMessage` and `displayWonGameMessage` to include additional output message (**I.**

		missing parameter if `NoOfMoves` declared as global)
		
		`void displayWonGameMessage(int noOfMoves){`
		`console.println("ARGHHHHHH! The monster has eaten you. GAME OVER.");`
		`console.println("Maybe you will have better luck next time you play MONSTER!");`
		`console.println("The number of moves you survived in the cavern was " + noOfMoves);`
		`console.println();`
		`}`
		
		`void displayWonGameMessage(int noOfMoves){`
		`console.println("Well done! You have found the flask containing the Styxian potion.");`
		`console.println("You have won the game of MONSTER!");`
		`console.println("The number of moves you took to find the flask was " + noOfMoves);`
		`}`
			5

11	42	`int calculateDistance(CellReference playerPosition, CellReference monsterPosition) {`
		`int distance;`
		
		`if(playerPosition.noOfCellsEast>monsterPosition.noOfCellsEast){`
		`distance=playerPosition.noOfCellsEast-monsterPosition.noOfCellsEast;`
		`} else{`
		`distance=monsterPosition.noOfCellsEast-playerPosition.noOfCellsEast;`
		`}`
		
		`if(playerPosition.noOfCellsSouth>monsterPosition.noOfCellsSouth){`
		
		`distance=distance+playerPosition.noOfCellsSouth-monsterPosition.noOfCellsSouth;`
		`}else{`
		
		`distance=distance+monsterPosition.noOfCellsSouth-playerPosition.noOfCellsSouth;`
		`}`
		`return distance;`
		`}`
		
		Alternative Answer
		`int calculateDistance(CellReference playerPosition, CellReference monsterPosition) {`	

```
  int distance;
  distance =
Math.abs(playerPosition.noOfCellsSouth
- monsterPosition.noOfCellsSouth);
  distance +=
Math.abs(playerPosition.noOfCellsEast -
monsterPosition.noOfCellsEast);
  return distance;
}
```

Alternative Answer

```
distance=(int)Math.sqrt(Math.pow((double)(playerPo
sition.noOfCellsSouth -
monsterPosition.noOfCellsSouth), 2))

+(int)Math.sqrt(Math.pow((double)(playerPosition.n
oOfCellsEast - monsterPosition.noOfCellsEast),
2));
```

Alternative Answer

```
distance=(int)Math.round(Math.sqrt(Math.pow((doubl
e)(playerPosition.noOfCellsSouth -
monsterPosition.noOfCellsSouth), 2))

+Math.sqrt(Math.pow((double)(playerPosition.noOfCe
llsEast - monsterPosition.noOfCellsEast), 2)));
```

Alternative answer

```
int distance2;
...
distance = playerPosition.noOfCellsEast -
monsterPosition.noOfCellsEast;
if (distance < 0) {
  distance = distance * -1;
}
distance2 = playerPosition.noOfCellsSouth -
monsterPosition.noOfCellsSouth;
if (distance2 < 0) {
  distance2 = distance2 * -1;
}
distance = distance + distance2;
```

7

	43	displayMoveOptions(); **console.println("Distance between monster and player: " + calculateDistance(playerPosition, monsterPosition));**	**3**

PYTHON Mark Scheme

Qu	Part	Marking Guidance	Marks

6	16	**# Section B Q6 Python 2.6**
		`Answer = 0`
		`Bit = 0`
		`Column = 8`
		
		`while Column >= 1:`
		`  print "Enter bit value: "`
		`  # Accept: Bit = int(raw_input("Enter bit value: "))`
		`  Bit = input()`
		`  Answer = Answer + (Column * Bit)`
		`  Column = Column // 2`
		`print "Decimal value is: ", Answer`
		`# or + str(Answer)`
		
		# Section B Q6 Python 3.1
		`Answer = 0`
		`Bit = 0`
		`Column = 8`
		
		`while Column >= 1:`
		`  print("Enter bit value: ")`
		`  # Accept: Bit = int(input("Enter bit value: "))`
		`  Bit = int(input())`
		`  Answer = Answer + (Column * Bit)`
		`  Column = Column // 2`
		`print("Decimal value is: " + str(Answer))`
		`# or print("Decimal value is: {0}".format(Answer))`
		
		A. `Answer` and `Bit` not declared at start as long as they are spelt correctly and when they are given an initial value that value is of the correct data type
			11

8	32	**Python 2**
		`def DisplayMoveOptions():`
		`  print ''`
		`  print 'Enter N to move NORTH'`
		`  print 'Enter E to move EAST'`
		`  print 'Enter S to move SOUTH'`
		`  print 'Enter W to move WEST'`
		`  **print 'Enter D to move SOUTHEAST'**`
		`  print 'Enter M to return to the Main Menu'`
		`  print ''`
		
		Python 3
		`def DisplayMoveOptions():`
		`  print ()`
		`  print ('Enter N to move NORTH')`
		`  print ('Enter E to move EAST')`
		`  print ('Enter S to move SOUTH')`	

		`  print ('Enter W to move WEST')`
		`  print ('Enter D to move SOUTHEAST')`
		`  print ('Enter M to return to the Main Menu')`
		`  print ()`
			1

	33	`elif Direction == 'E':`
		`  PlayerPosition.NoOfCellsEast += 1`
		`elif Direction == 'D':`
		`  PlayerPosition.NoOfCellsSouth += 1`
		`  PlayerPosition.NoOfCellsEast += 1`
			3

	34	`def CheckValidMove(PlayerPosition,Direction):`
		`  ValidMove = True`
		`  if not (Direction in ['N','S','W','E','D','M']):`
		`  ValidMove = False`
		`  return ValidMove`
			1

9	36	`while not ValidMove:`
		`  DisplayMoveOptions()`
		`  MoveDirection = GetMove()`
		`  ValidMove = CheckValidMove(PlayerPosition, MoveDirection)`
		`  if not ValidMove:`
		`  # Python 2:`
		`  print 'That is not a valid move, please try again'`
		`  # Python 3:`
		`  print('That is not a valid move, please try again')`
		
		Alternative answer
		`if ValidMove = False...`
			3

	37	`def CheckValidMove(PlayerPosition,Direction):`
		`  ValidMove = True`
		`  if not (Direction in ['N','S','W','E','D','M']):`
		`  ValidMove = False`
		`  if (PlayerPosition.NoOfCellsSouth == 1) and (Direction == 'N'):`
		`  ValidMove = False`
		`  return ValidMove`
		
		Alternative answer
		`if not (Direction in ['N','S','W','E','D','M']):`
		`  ValidMove = False`
		`if ValidMove and (Direction == 'N'):`
		`  ValidMove = (ValidMove and (PlayerPosition.NoOfCellsSouth != 1))`
			4

10	39	Eaten = False
		FlaskFound = False
		MoveDirection = ''
		DisplayCavern(Cavern, MonsterAwake)
		NoOfMoves = 0
		while not (Eaten or FlaskFound or (MoveDirection == 'M')):
		  ValidMove = False
		  while not ValidMove:
		  DisplayMoveOptions()
		  MoveDirection = GetMove()
		  ValidMove = CheckValidMove(PlayerPosition, MoveDirection)
		  if not ValidMove:
		  print 'That is not a valid move, please try again'
		  if MoveDirection != 'M':
		  MakeMove(Cavern, MoveDirection, PlayerPosition)
		  NoOfMoves += 1
		  DisplayCavern(Cavern, MonsterAwake)
		...
		  if FlaskFound:
		  DisplayWonGameMessage()
		  # Python 2:
		  print 'The number of moves you took to find the flask was', NoOfMoves
		  # Alternative answer:
		  print 'The number of moves you took to find the flask was ' + str(NoOfMoves)
		  # Python 3:
		  print('The number of moves you took to find the flask was ' + str(NoOfMoves)
		  # Alternative answer:
		  print('The number of moves you took to find the flask was {0}'.format(NoOfMoves)) #Py3
		
		...
		if Eaten:
		  DisplayLostGameMessage()
		  # Python 2:
		  print 'The number of moves that you survived in the cavern for was', NoOfMoves
		  # Alternative answer:
		  print 'The number of moves that you survived in the cavern for was ' + str(NoOfMoves)
		  # Python 3:
		  print('The number of moves that you survived in the cavern for was ' + str(NoOfMoves))
		  # Alternative answer:
		  print('The number of moves that you survived in the cavern for was {0}'.format(NoOfMoves))	

Alternative Answer

```
# Python 2
if Eaten:
  print 'The number of moves that you survived in the
cavern for was', NoOfMoves
else:
  print 'The number of moves you took to find the
flask was', NoOfMoves

# Python 3
if Eaten:
  print('The number of moves that you survived in the
cavern for was' + str(NoOfMoves))
else:
  print('The number of moves you took to find the
flask was' + str(NoOfMoves))
```

A. `.format(NoOfMoves)`

Alternative answer

```
if FlaskFound:
  DisplayWonGameMessage(NoOfMoves)
...
if Eaten:
  DisplayLostGameMessage(NoOfMoves)
```

together with modified `displayLostGameMessage` and `displayWonGameMessage` to include additional output message (**I.** missing parameter if `NoOfMoves` declared as global)

 # Python 2

```
def DisplayWonGameMessage(NoOfMoves):
  print 'Well Done!  You have found the flask
containing the Styxian potion.'
  print 'You have won the game of MONSTER!'
  print 'The number of moves you took to find the
flask was ', NoOfMoves

def DisplayLostGameMessage(NoOfMoves):
  print 'ARGHHHHHH!  The monster has eaten you.  GAME
OVER.'
  print 'Maybe you will have better luck the next
time you play MONSTER!'
  print 'The number of moves that you survived in the
cavern for was', NoOfMoves
```

 # Python 3

```
def DisplayWonGameMessage(NoOfMoves):
  print('Well Done!  You have found the flask
containing the Styxian potion.')
```

```
  print('You have won the game of MONSTER!')
  print('The number of moves you took to find the
flask was' + str(NoOfMoves))

def DisplayLostGameMessage(NoOfMoves):
  print('ARGHHHHHH!  The monster has eaten you.  GAME
OVER.')
  print('Maybe you will have better luck the next
time you play MONSTER!')
print('The number of moves that you survived in the
cavern for was'+ str(NoOfMoves))
```

5

11	42	**Alternative Answer** **Alternative Answer** **Alternative Answer**

```
def CalculateDistance(PlayerPosition,
MonsterPosition):
  if PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast:
    Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
  else:
    Distance = MonsterPositionNoOfCellsEast -
PlayerPosition.NoOfCellsEast
  if PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth:
    Distance = Distance +
PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
  else:
    Distance = Distance +
MonsterPositionNoOfCellsSouth -
PlayerPosition.NoOfCellsSouth
  return Distance
```

```
Distance = abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

```
return abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

```
import math
Distance =
math.trunc(math.sqrt(pow((PlayerPosition.NoOfCellsEas
t - MonsterPosition.NoOfCellsEast),2)) +
math.sqrt(pow((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth),2)))
```

		Alternative Answer
		`import math`
		`Distance =`
		`round(math.sqrt((PlayerPosition.NoOfCellsEast -`
		`MonsterPosition.NoOfCellsEast)**2) +`
		`math.sqrt((PlayerPosition.NoOfCellsSouth -`
		`MonsterPosition.NoOfCellsSouth)**2))`
		
		Alternative Answer
		`Distance = PlayerPosition.NoOfCellsEast -`
		`MonsterPosition.NoOfCellsEast`
		`if Distance < 0:`
		`  Distance = Distance * -1`
		`Distance2 = PlayerPosition.NoOfCellsSouth -`
		`MonsterPosition.NoOfCellsSouth`
		`if Distance2 < 0:`
		`  Distance2 = Distance2 * -1`
		`Distance = Distance + Distance2`
			7

	43	**`# Python 2:`** `DisplayMoveOptions()` **`print 'Distance to monster:',`** **`CalculateDistance(PlayerPosition, MonsterPosition)`** **`# Alternative answer:`** `DisplayMoveOptions()` **`print 'Distance to monster:' +`** **`str(CalculateDistance(PlayerPosition,`** **`MonsterPosition))`** **`# Python 3:`** `DisplayMoveOptions()` **`print('Distance to monster:' +`** **`str(CalculateDistance(PlayerPosition,`** **`MonsterPosition))`**	**3**

UMS conversion calculator www.aqa.org.uk/umconversion