

OXFORD

INTERNATIONAL
AQA EXAMINATIONS

INTERNATIONAL GCSE

COMPUTER SCIENCE

9210

Paper 1 Programming

Mark scheme

November 2020

Version: 1.0 Final



2 0 B Y 9 2 1 0 1 / M S

Mark schemes are prepared by the Lead Assessment Writer 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 associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students' scripts. Alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' 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 oxfordaqaexams.org.uk

Copyright information

OxfordAQA retains the copyright on all its publications. However, registered schools/colleges for OxfordAQA are permitted to copy material from this booklet for their own internal use, with the following important exception: OxfordAQA cannot give permission to schools/colleges to photocopy any material that is acknowledged to a third party even for internal use within the centre.

Copyright © 2020 Oxford International AQA Examinations and its licensors. All rights reserved.

Question	Part	Marking guidance	Total marks
01	1	Hide; GameWon; Max 1	1 AO3=1
01	2	CheckCharacter; 	1 AO3=1
01	3	Improves readability of code; Easier to update the programming code if the value changes; A. by implication Reduce the likelihood of errors; Unlike a variable the value can't be (accidentally) changed (during program execution); Max 2	2 AO3=2
01	4	When the value is passed to the subroutine/CharacterSymbol is not an upper case character/letter; and not a lower case character/letter; and not a digit (character); A. character or letter for 1 mark	3 AO3=3
01	5	Makes structure of program clearer // easier to understand; Subroutines can be easily re-used; If a subroutine is called more than once, it will reduce the memory required by a program; Subroutines can be tested independently; Subroutines can be developed by different members of a team; Max 3	3 AO3=3
01	6	2; 	1 AO3=1

01	7	An integer variable stores whole numbers//stores numbers without a decimal/fractional part; A real variable stores numbers with a decimal/fractional part;	2 AO3=2
01	8	Checking the content of the cells (vertically) next to the cell location passed to the subroutine; If cell checked contains a lower case letter/treasure then treasure count is increased (by one)/counts treasure; Only checks one side of the cell location if cell is on a vertical edge; Max 2	2 AO3=2
01	9	It checks if the cell selected is not <code>EMPTY</code> / <code>EMPTY</code> ; while it is not <code>EMPTY</code> the coordinates of the cell are randomly generated; Or The purpose is to find an empty cell; so an item can be stored in this cell;	2 AO3=2
01	10	Local variables can be accessed/used only within subroutine they are declared in // Global variables can be accessed anywhere in a program; Local variables only use memory when the subroutine is running // Global variable use memory for the whole time the program is running;	2 AO3=2

Question	Part	Marking guidance	Total marks
02	1	Set <code>StartRow</code> to 0;	1 AO3=1
02	2	Set <code>EndRow</code> to 1;	1 AO3=1
02	3	Is <code>Column</code> = 0?;	1 AO3=1

Question	Part	Marking guidance	Total marks
03	1	1 mark: Program displays correct message. A. minor typos and incorrect case. I. full stop at end of message. 1 mark: Statement is at a position so that message is displayed at the start of each game and before any other output from the <code>PlayGame</code> subroutine. 1 mark: Blank line is printed after the message is displayed. Python <pre>def PlayGame(Board, BoardDimension): Hide = True GameWon = False Moves = 0 MaxMoves = SetMaxMoves(BoardDimension) ScoreToWin = SetScoreToWin(BoardDimension) print("Welcome to Treasure Hunt ") print() print("You have to score " + str(ScoreToWin) + " points to win.") print("You can make a maximum of " + str(MaxMoves) + " moves.") while not GameWon and Moves < MaxMoves:</pre> Visual Basic <pre>Sub PlayGame(Board As Char(), BoardDimension As Integer) Dim Hide As Boolean = True Dim GameWon As Boolean = False Dim Moves Integer = 0 Dim MaxMoves As Integer = SetMaxMoves(BoardDimension) Dim ScoreToWin As Integer = SetScoreToWin(BoardDimension) Console.WriteLine("Welcome to Treasure Hunt ") Console.WriteLine() Console.WriteLine("You have to score " & ScoreToWin & " points to win.") Console.WriteLine("You can make a maximum of " & MaxMoves & " moves.") While Not GameWon And Moves < MaxMoves DisplayBoard(Board, BoardDimension, Hide) Dim Column, Row As Integer GetColumnRow(BoardDimension, Column, Row) Board = MakeMove(Column, Row, Board, BoardDimension) Moves = Move + 1 Dim Score As Integer = CalculateScore(Board, BoardDimension) If Score >= ScoreToWin Then GameWon = True End If End While DisplayBoard(Board, BoardDimension, Hide)</pre>	3 AO3=3

		<pre> Console.WriteLine() If GameWon Then Console.WriteLine("You won the game in " + Moves + " moves.") Else Console.WriteLine("You have lost the game as you have made " + Moves + " moves.") End If Console.WriteLine() End Sub C# private static void PlayGame(char[,] Board, int BoardDimension) { bool Hide = true; bool GameWon = false; int Moves = 0; int MaxMoves = SetMaxMoves(BoardDimension); int ScoreToWin = SetScoreToWin(BoardDimension); Console.WriteLine("Welcome to Treasure Hunt "); Console.WriteLine(); Console.WriteLine("You have to score " + ScoreToWin + " points to win."); Console.WriteLine("You can make a maximum of " + MaxMoves + " moves."); while (!GameWon && Moves < MaxMoves) { DisplayBoard(Board, BoardDimension, Hide); int Column = 0, Row = 0; GetColumnRow(BoardDimension, ref Column, ref Row); Board = MakeMove(Column, Row, Board, BoardDimension); Moves = Moves + 1; int Score = CalculateScore(Board, BoardDimension); if (Score >= ScoreToWin) { GameWon = true; } } DisplayBoard(Board, BoardDimension, Hide); Console.WriteLine(); if (GameWon) { Console.WriteLine("You won the game in " + Moves + " moves."); } else { Console.WriteLine("You have lost the game as you have made " + Moves + " moves."); } Console.WriteLine(); } </pre>	
--	--	---	--

03	2	Screen capture(s) must match code from 03.1 1 mark: Correct message displayed each time the game is played. A. incorrect message and missing blank line if it matches code in 03.1 <pre> MAIN MENU 1. Play sample game 2. Play easy game 3. Play hard game 9. Quit Please enter your choice: 1 Welcome to Treasure Hunt You have to score 9 points to win. You can make a maximum of 17 moves.</pre>	1 AO3=1
----	---	---	-------------------------------------

Question	Part	Marking guidance	Total marks
04	1	1 mark: Code added in the correct place in the PlayGame subroutine. 1 mark: Program displays correct message. A. other similar suitable message 1 mark: New local variable is created and used to store value input following message. A. if value input is used in the if statement and not saved in a variable 1 mark: If statement created that is used to check value input 1 mark: Hide set to false if and only if the input equals Y. Python <pre>def PlayGame(Board, BoardDimension): Hide = True GameWon = False Moves = 0 MaxMoves = SetMaxMoves(BoardDimension) ScoreToWin = SetScoreToWin(BoardDimension) print("You have to score " + str(ScoreToWin) + " points to win.") print("You can make a maximum of " + str(MaxMoves) + " moves.") Show = input("Enter Y to show where the treasure is hidden: ") if Show == "Y": Hide = False while not GameWon and Moves < MaxMoves: DisplayBoard(Board, BoardDimension, Hide) Column, Row = GetColumnRow(BoardDimension)</pre> Visual Basic <pre>Console.WriteLine("You can make a maximum of " & MaxMoves & " moves.") Console.WriteLine("Enter Y to show where the treasure is hidden: ") Dim Show As String = Console.ReadLine() If Show = "Y" Then Hide = False End If While Not GameWon And Moves < MaxMoves</pre> C# <pre>Console.WriteLine("You can make a maximum of " + MaxMoves + " moves."); Console.WriteLine("Enter Y to show where the treasure is hidden: ");</pre>	5 AO3=5

		<pre>string Show = Console.ReadLine(); if (Show == "Y") { Hide = false; } while (!GameWon && Moves < MaxMoves)</pre>	
04	2	Screen capture(s) must match code from 04.1 1 mark: Suitable prompt displayed matching code from 4.1 and grid is displayed showing where treasure is located after Y input. Enter Y to show where the treasure is hidden: Y The treasure map looks like this: <pre> 0 1 2 3 4 0 g g . . . 1 g . g g . 2 . g g g . 3 . . g . . 4 . . . g .</pre>	1 AO3=1

Question	Part	Marking guidance	Total marks
05	1	1 mark: Points required to win the game is calculated correctly after call to <code>CalculateScore</code> subroutine in an appropriate place. 1 mark: A check for player not winning the game is made in the while loop // use of an else with the appropriate if. A. the message displayed at the start of the while loop before the call to <code>DisplayBoard</code> and score is now initialised to 0 outside the while loop. 1 mark: The message with the correct value for points required to win the game is only displayed if the game is not won each time a move is made. A. the use of the value they have calculated. Python <pre>def PlayGame(Board, BoardDimension): Hide = True GameWon = False Moves = 0 MaxMoves = SetMaxMoves(BoardDimension) ScoreToWin = SetScoreToWin(BoardDimension) print("You have to score " + str(ScoreToWin) + " points to win.") print("You can make a maximum of " + str(MaxMoves) + " moves.") while not GameWon and Moves < MaxMoves: DisplayBoard(Board, BoardDimension, Hide) Column, Row = GetColumnRow(BoardDimension) Board = MakeMove(Column, Row, Board, BoardDimension) Moves = Moves + 1 Score = CalculateScore(Board, BoardDimension) if Score >= ScoreToWin: GameWon = True else: print("You need to score", ScoreToWin - Score, "more points to win the game.") DisplayBoard(Board, BoardDimension, Hide)</pre> Alternative Python <pre>..... Score = CalculateScore(Board, BoardDimension)</pre>	3 AO3=3

		<pre> if Score >= ScoreToWin: GameWon = True If GameWon == False: print("You need to score", ScoreToWin - Score, "more points to win the game.") DisplayBoard(Board, BoardDimension, Hide) </pre> Visual Basic <pre> Dim Score As Integer = CalculateScore(Board, BoardDimension) If Score >= ScoreToWin Then GameWon = True Else Console.WriteLine("You need to score " & ScoreToWin - Score & " more points to win the game.") End If End While </pre> C# <pre> int Score = CalculateScore(Board, BoardDimension); if (Score >= ScoreToWin) { GameWon = true; } else { Console.WriteLine("You need to score " + (ScoreToWin - Score) + " more points to win the game."); } </pre>	
--	--	--	--

05	2	Screen capture(s) must match code from 05.1 1 mark: Suitable message is displayed showing points required to win the game. A. if points not calculated correctly but matches their code in 07.1 MAIN MENU 1. Play sample game 2. Play easy game 3. Play hard game 9. Quit Please enter your choice: 1 You have to score 9 points to win. You can make a maximum of 17 moves. The treasure map looks like this: <pre> 0 1 2 3 4 0 1 2 3 4 </pre> Please enter column (between 0 and 4): 0 Please enter row (between 0 and 4): 0 Well done! The cell (0,0) contains treasure. You need to score 8 more points to win the game. The treasure map looks like this:	1 AO3=1
----	---	--	-------------------------------------

Question	Part	Marking guidance	Total marks
06	1	1 mark: Loop added in an appropriate place in <code>GetColumnRow</code> subroutine. 1 mark: Check against lower/upper boundary for <code>Column</code>. 1 mark: Check against other boundary. 1 mark: Correctly combine the two checks. 1 mark: The message <code>Invalid column</code> is displayed only if invalid value has been entered for at least one of the correct bounds. I. case of output message. A. minor typos in output message. I. spacing in output message. 1 mark: New value for <code>Column</code> is input and stored correctly inside the loop. Python <pre>def GetColumnRow(BoardDimension): print() Column = int(input("Please enter column (between 0 and " + str(BoardDimension - 1) +"): ")) while Column < 0 or Column > BoardDimension - 1: print("Invalid column") Column = int(input("Please enter column (between 0 and " + str(BoardDimension - 1) +"): ")) Row = int(input("Please enter row (between 0 and " + str(BoardDimension - 1) +"): ")) print() return Column, Row</pre> Alternative Answers <pre>while Column not in range(0, BoardDimension): print("Invalid column") Column = int(input("Please enter column (between 0 and " + str(BoardDimension - 1) +"): ")) while not(0 <= Column <= BoardDimension - 1): print("Invalid column") Column = int(input("Please enter column (between 0 and " + str(BoardDimension - 1) +"): "))</pre> Visual Basic	6 AO3=6

		<pre> Sub GetColumnRow(BoardDimension As Integer, ByRef Column As Integer, ByRef Row As Integer) Console.WriteLine() Console.Write("Please enter column (between 0 and " & (BoardDimension - 1) & "): ") Column = Console.ReadLine() While Column < 0 Or Column > BoardDimension - 1 Console.WriteLine("Invalid Column") Console.Write("Please enter column (between 0 and " & (BoardDimension - 1) & "): ") Column = Console.ReadLine() End While Console.Write("Please enter row (between 0 and " & (BoardDimension - 1) & "): ") Row = Console.ReadLine() Console.WriteLine() End Sub </pre> C# <pre> private static void GetColumnRow(int BoardDimension, ref int Column, ref int Row) { Console.WriteLine(); Console.Write("Please enter column (between 0 and " + (BoardDimension - 1) + "): "); Column = int.Parse(Console.ReadLine()); while (Column < 0 Column > BoardDimension - 1) { Console.WriteLine("Invalid Column"); Console.Write("Please enter column (between 0 and " + (BoardDimension - 1) + "): "); Column = int.Parse(Console.ReadLine()); } Console.Write("Please enter row (between 0 and " + (BoardDimension - 1) + "): "); Row = int.Parse(Console.ReadLine()); Console.WriteLine(); } </pre>	
--	--	---	--

06	2	Screen capture(s) must match code from 06.1 1 mark: Correct error message is displayed after invalid column value input testing lower bound (-1) and prompt for column displayed. 1 mark: Correct error message is displayed after invalid column value input testing upper bound (5) and prompt for column displayed. A. incorrect message if it matches code in 06.1 <pre> 0 1 2 3 4 0 1 2 3 4 Please enter column (between 0 and 4): -1 Invalid column Please enter column (between 0 and 4): 5 Invalid column Please enter column (between 0 and 4): 1 Please enter row (between 0 and 4): </pre>	2 AO3=2
----	---	---	-------------------------------------

Question	Part	Marking guidance	Total marks
07	1	1 mark: If statement using a correct comparison with BoardDimension. 1 mark: Correct lower limit for one board size (set and stored in an appropriate variable). 1 mark: Correct upper limit for one board size (set and stored in an appropriate variable). 1 mark: Correct use of ELSE // second selection structure 1 mark: Correct lower and upper limit for other board size (set and stored in an appropriate variable). 1 mark: Random number generated using upper and lower limit. A. incorrect upper and lower limits 1 mark: Correct range of random numbers can be generated. 1 mark: The random value the code generates is always returned. Max 7: if code contains errors Python <pre>def SetMaxMoves(BoardDimension): if BoardDimension < 10: Lower = 3 * BoardDimension Upper = 4 * BoardDimension else: Lower = 5 * BoardDimension Upper = 6 * BoardDimension MaxMoves = random.randint(Lower, Upper) return MaxMoves</pre> Visual Basic <pre>Function SetMaxMoves(BoardDimension As Integer) As Integer Dim MaxMoves As Integer Dim Lower, Upper As Integer If BoardDimension < 10 Then Lower = 3 * BoardDimension Upper = 4 * BoardDimension Else Lower = 5 * BoardDimension Upper = 6 * BoardDimension End If MaxMoves = RandomGenerator.Next(Lower, Upper + 1) Return MaxMoves End Function</pre> C# <pre>private static int SetMaxMoves(int BoardDimension) { int MaxMoves; int Lower, Upper;</pre>	8 AO3=8

		<pre> if (BoardDimension < 10) { Lower = 3 * BoardDimension; Upper = 4 * BoardDimension; } else { Lower = 5 * BoardDimension; Upper = 6 * BoardDimension; } MaxMoves = RandomGenerator.Next(Lower, Upper + 1); return MaxMoves; } </pre>	
--	--	---	--

07	2	Screen capture(s) must match code from 07.1 1 mark: Hard game chosen and maximum moves allowed is between 50 and 60 MAIN MENU <pre> 1. Play sample game 2. Play easy game 3. Play hard game 9. Quit </pre> Please enter your choice: 3 You have to score 19 points to win. You can make a maximum of 54 moves. The treasure map looks like this:	1 AO3=1
-----------	----------	---	-------------------------------------

Question	Part	Marking guidance	Total marks
08	1	1 mark: Half the treasure items placed randomly on the board are gold coins. 1 mark: Half the treasure items placed randomly on the board are diamonds. 1 mark: <code>PlaceItem</code> subroutine is called correctly when placing a diamond by passing <code>d</code> as the argument for <code>Item</code>. A. their representation of a diamond instead of <code>d</code>. 1 mark: The correct <code>Board</code> is returned under all circumstances Python <pre>def InitialiseRandomBoard(Board, BoardDimension): if BoardDimension >= 10: NumberOfTreasureItems = BoardDimension * 6 else: NumberOfTreasureItems = BoardDimension * 2 for Count in range(NumberOfTreasureItems//2): Board = PlaceItem(Board, BoardDimension, GOLDCOIN) Board = PlaceItem(Board, BoardDimension, "d") return Board</pre> Visual Basic <pre>Sub InitialiseRandomBoard(Board As Char(), BoardDimension As Integer) Dim NumberOfTreasureItems As Integer If BoardDimension >= 10 Then NumberOfTreasureItems = BoardDimension * 6 Else NumberOfTreasureItems = BoardDimension * 2 End If For Count = 1 To NumberOfTreasureItems \ 2 PlaceItem(Board, BoardDimension, GoldCoin) PlaceItem(Board, BoardDimension, "d") Next End Sub</pre> C# <pre>private static void InitialiseRandomBoard(char[,] Board, int BoardDimension) { int NumberOfTreasureItems; if (BoardDimension >= 10) { NumberOfTreasureItems = BoardDimension * 6; } else { NumberOfTreasureItems = BoardDimension * 2;</pre>	4 AO3=4

		<pre> } for (int Count = 0; Count < NumberOfTreasureItems / 2; Count++) { PlaceItem(Board, BoardDimension, GoldCoin); PlaceItem(Board, BoardDimension, 'd'); } } </pre>	
--	--	--	--

08	2	1 mark: Selection structure added to <code>CalculateScore</code> subroutine to check if current location contains a <code>D</code> // Amendment of current selection structure in <code>CalculateScore</code> so that it now checks if current location contains a <code>D</code>. 1 mark: Selection structure for a <code>D</code> being found 1 mark: If a <code>D</code> found then <code>Score</code> is increased by 2 1 mark: Variable created for counting the diamonds found. 1 mark: Variable created for counting diamonds found is initialised to 0. 1 mark: Diamonds found counted correctly. 1 mark: Suitable message containing number of diamonds found is displayed in the correct location. 1 mark: Variable created for counting the diamonds not found. 1 mark: Variable created for counting diamonds not found is initialised to 0. 1 mark: Diamonds not found counted correctly. 1 mark: If all diamonds have been found then <code>Score</code> is increased by 5 1 mark: Correct value for <code>Score</code> always returned. A their representation of a diamond that has been found instead of <code>D</code> Python <pre> def CalculateScore(Board, BoardDimension): Score = 0 DiamondFoundCount = 0 DiamondNotFoundCount = 0 for Row in range(BoardDimension): for Column in range(BoardDimension): if Board[Row][Column] == GOLDCOIN.upper(): Score = Score + 1 if Board[Row][Column] == "D": Score = Score + 2 DiamondFoundCount = DiamondFoundCount + 1 if Board[Row][Column] == "d": DiamondNotFoundCount = DiamondNotFoundCount + 1 </pre>	12 AO3=12
-----------	----------	--	---------------------------------------

		<pre> if DiamondNotFoundCount == 0: Score = Score + 5 print("The number of diamonds found so far is", DiamondFoundCount) return Score </pre> Visual Basic <pre> Function CalculateScore(Board As Char(), BoardDimension As Integer) As Integer Dim Score As Integer = 0 Dim DiamondFoundCount = 0 Dim DiamondNotFoundCount = 0 For Row = 0 To BoardDimension - 1 For Column = 0 To BoardDimension - 1 If Board(Row, Column) = GoldCoin.ToUpper() Then Score = Score + 1 End If If Board(Row, Column) = "D" Then Score = Score + 2 DiamondFoundCount = DiamondFoundCount + 1 End If If Board(Row, Column) = "d" Then DiamondNotFoundCount = DiamondNotFoundCount + 1 End If Next Next If DiamondNotFoundCount = 0 Then Score = Score + 5 End If Console.WriteLine("The number of diamonds found so far is " & DiamondFoundCount) Return Score End Function </pre> C# <pre> private static int CalculateScore(char[,] Board, int BoardDimension) { int Score = 0; int DiamondFoundCount = 0; int DiamondNotFoundCount = 0; for (int Row = 0; Row < BoardDimension; Row++) { for (int Column = 0; Column < BoardDimension; Column++) { if (Board[Row, Column] == char.ToUpper(GoldCoin)) { Score = Score + 1; } if (Board[Row, Column] == 'D') { Score = Score + 2; DiamondFoundCount = DiamondFoundCount + 1; } if (Board[Row, Column] == 'd') { </pre>	
--	--	---	--

		<pre> DiamondNotFoundCount = DiamondNotFoundCount + 1; } } if (DiamondNotFoundCount == 0) { Score = Score + 5; } Console.WriteLine("The number of diamonds found so far is " + DiamondFoundCount); return Score; } </pre>	
--	--	---	--

08	3	Screen capture(s) must match code from 08.1 and 08.2 1 mark: A grid is displayed showing a D in the location the candidate has entered along with a suitable message regarding the number of diamonds found so far // a grid showing d displayed in the location the candidate goes on to enter along with a suitable messages regarding the number of diamonds found so far. A. their representation of a diamond that has been found. Well done! The cell (3,3) contains treasure. The number of diamonds found so far is 1 The treasure map looks like this: <pre> 0 1 2 3 4 0 1 . 0 . . . 2 . . 1 . . 3 . . . D . 4 </pre>	1 AO3=1
-----------	----------	---	-------------------------------------

Question	Part	Marking guidance	Total marks
09	1	1 mark: Use of nested loop // use of 4 if statements A. alternative mechanism which would be appropriate if implemented correctly. 1 mark: Two diagonal neighbouring cells checked. I if it does not work correctly for empty cell selected on edge of board. 1 mark: Treasure count increment by 1 if treasure found in two checks for diagonal neighbours. 1 mark: Other two diagonal neighbouring cells checked. I if it does not work correctly for empty cell selected on edge of board. 1 mark: Treasure count increment by 1 if treasure found in other two checks for diagonal neighbours. 1 mark: Check works correctly on two of the edges when check for diagonal neighbour made. 1 mark: Check works correctly for other two edges when check for diagonal neighbour made. 1 mark: Correct value for neighbouring cells containing treasure is returned by subroutine. Max 6 if horizontal or vertical checks no longer work correctly Python <pre>def CheckSurroundingCells (Board, BoardDimension, Row, Column): Neighbouring = 0 StartRow, EndRow, StartColumn, EndColumn = CheckRange(BoardDimension, Row, Column) for R in range(StartRow, EndRow + 1): for C in range(StartColumn, EndColumn + 1): if R != Row or C != Column: #not required CharacterRange = CheckCharacter(Board[R][C]) if CharacterRange == "Lower Case": Neighbouring = Neighbouring + 1 return Neighbouring</pre> Alternative Answer <pre>def CheckSurroundingCells(Board, BoardDimension, Row, Column): Neighbouring = 0 StartRow, EndRow, StartColumn, EndColumn = CheckRange(BoardDimension, Row, Column)</pre>	8 AO3=8

		<pre> if Row != 0 and Column != 0 and CheckCharacter(Board[StartRow][StartColumn]) == "Lower Case": Neighbouring = Neighbouring + 1 if Row != BoardDimension - 1 and Column != 0 and CheckCharacter(Board[EndRow][StartColumn]) == "Lower Case": Neighbouring = Neighbouring + 1 if Row != 0 and Column != BoardDimension - 1 and CheckCharacter(Board[StartRow][EndColumn]) == "Lower Case": Neighbouring = Neighbouring + 1 if Row != BoardDimension - 1 and Column != BoardDimension - 1 and CheckCharacter(Board[EndRow][EndColumn]) == "Lower Case": Neighbouring = Neighbouring + 1 for R in range(StartRow, EndRow + 1): if R != Row: CharacterRange = CheckCharacter(Board[R][Column]) if CharacterRange == "Lower Case": Neighbouring = Neighbouring + 1 for C in range(StartColumn, EndColumn + 1): if C != Column: CharacterRange = CheckCharacter(Board[Row][C]) if CharacterRange == "Lower Case": Neighbouring = Neighbouring + 1 return Neighbouring </pre> Visual Basic <pre> Function CheckSurroundingCells(Board As Char(,), BoardDimension As Integer, Row As Integer, Column As Integer) As Integer Dim Neighbouring As Integer = 0 Dim StartRow, EndRow, StartColumn, EndColumn As Integer Dim CharacterRange As String CheckRange(BoardDimension, Row, Column, StartRow, EndRow, StartColumn, EndColumn) If Row <> 0 And Column <> 0 And CheckCharacter(Board(StartRow, StartColumn)) = "Lower Case" Then Neighbouring = Neighbouring + 1 End If If Row <> (BoardDimension - 1) And Column <> 0 And CheckCharacter(Board(EndRow, StartColumn)) = "Lower Case" Then Neighbouring = Neighbouring + 1 End If If Row <> 0 And Column <> (BoardDimension - 1) And CheckCharacter(Board(StartRow, EndColumn)) = "Lower Case" Then Neighbouring = Neighbouring + 1 End If </pre>	
--	--	--	--

		<pre> If Row <> (BoardDimension - 1) And Column <> (BoardDimension - 1) And CheckCharacter(Board(EndRow, EndColumn)) = "Lower Case" Then Neighbouring = Neighbouring + 1 End If For R = StartRow To EndRow </pre> C# <pre> private static int CheckSurroundingCells(char[,] Board, int BoardDimension, int Row, int Column) { int Neighbouring = 0; int StartRow = 0, EndRow = 0, StartColumn = 0, EndColumn = 0; string CharacterRange; CheckRange(BoardDimension, Row, Column, ref StartRow, ref EndRow, ref StartColumn, ref EndColumn); if (Row != 0 && Column != 0 && CheckCharacter(Board[StartRow, StartColumn]) == "Lower Case") { Neighbouring = Neighbouring + 1; } if (Row != BoardDimension - 1 && Column != 0 && CheckCharacter(Board[EndRow, StartColumn]) == "Lower Case") { Neighbouring = Neighbouring + 1; } if (Row != 0 && Column != BoardDimension - 1 && CheckCharacter(Board[StartRow, EndColumn]) == "Lower Case") { Neighbouring = Neighbouring + 1; } if (Row != BoardDimension - 1 && Column != BoardDimension - 1 && CheckCharacter(Board[EndRow, EndColumn]) == "Lower Case") { Neighbouring = Neighbouring + 1; } for (int R = StartRow; R <= EndRow; R++) { </pre>	
--	--	--	--

09	2	Screen capture(s) must match code from 09.1 1 mark: Sample game chosen and treasure map displayed correctly when cell 1,1 entered. 1 mark: Treasure map displayed correctly when cell 2,0 entered. Please enter column (between 0 and 4): 1 Please enter row (between 0 and 4): 1 Number of moves made so far is 1 The treasure map looks like this: <pre> 0 1 2 3 4 0 1 . 6 . . . 2 3 4 </pre> Please enter column (between 0 and 4): 2 Please enter row (between 0 and 4): 0 The treasure map looks like this: <pre> 0 1 2 3 4 0 . . 3 . . 1 . 6 . . . 2 3 4 </pre> Please enter column (between 0 and 4):	2 AO3=2
----	---	---	-------------------------------------