

AS **Computing**

COMP1 – Problem Solving, Programming, Data Representation and Practical Exercise Mark scheme

2510 June 2016

Version :1.0 Final

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 Assessment Writer.

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 aqa.org.uk

Copyright © 2016 AQA and its licensors. All rights reserved.

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

Notation used in GCE Computing mark schemes:

- means a single mark
- ; // - means alternative response
- means an alternative word or sub-phrase /
- means acceptable creditworthy answer Α
- R - means reject answer as not creditworthy
- means ignore L
- means not enough NE

Qu	Part	Marking Guidance	Marks
1	01	180;	1
1	02	5.625 // 5 5/8 // 45/8 Mark as follows:	2
		3 bits before binary point correct (5); 5 bits after binary point correct (0.625 or 5/8); Alternative: 45/8;;	
1	03	Easier for people to read/understand; (Can be displayed using) fewer digits; More compact when printed/displayed; NE. Takes up less space NE. More compact MAX 1	1
1	04	B;4;	2
1	05	-;76;	2
1	06	4; I. any quotes	1
1	07	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 2	2
1	08	4;	1

2	09	The number of pixels/dots; per cm/inch/unit of measurement; //				ent;	2
		Width x height; in pixels	•				
2	10	2 ⁴ // 16;					1
2	11	512;;; // 16*16;*16;÷8; // 4096;; ÷8; MAX 2 if final answer not correct					
2	12	(For geometric images) NE. less space (For geometric images) (For geometric images) Can be scaled/resized/z scaled/resized/zoomed Image can be (more easily) manip Can preserve the backg deleted;	less stora will load f will down coomed w without lo sily) searc pulate ind round so	age space faster from load faster ithout disto ss of quali shed for pa ividual obj that it can	/memory likely secondary sto ;; ortion // can be ty; articular objects ects in an imag be recreated i	to be needed; rage; ;; je; f an object is	2
3	13	MAX 2					2
Ŭ	10	Origin	al State	Input	New State		2
		S	30	10	S40		
		S	30	20	S50		
		S	30	50	S0		
		S	30	R	S 0		
		Note: order of rows not	important	t			
		Mark as follows:					
		Any 2 rows correct;					
		All 3 rows correct;					
3	14	5;					1
4	15	12 (bits);					1
4	16	1600 // 800 * 2;					1
4	17	48000;;; // 12*1600;*20;/8;					3
		A. alternative values to	1600 only	if match (incorrect) ansv	ver from 16	

4	18	An analogue signal is an electrical signal;	3
		that represents analogue/continuous data // that varies in a continuous	
		manner.;	
		Digital signals are electrical signals;	
		(with voltage changes that are) in discrete steps.;	
		Note: Only one mark for stating that either of / both of analogue and	
		digital signals are electrical signals	

-			
5	19	1) Correct variable declarations for X, F, L and N ;	14
		Note for examiners	
		If a language allows variables to be used without explicit declaration (eg	
		Python) then this mark should be awarded if the four correct variables	
		exist in the program code and the first value they are assigned is of the	
		correct data type.	
		2) Correct output message Enter a number:;	
		3) N assigned the value entered by the user;	
		4) F assigned the value of 16.0;	
		5) IF statement with correct condition;	
		6) x assigned the value stored in N;	
		7) WHILE loop, with syntax allowed by the programming language with	
		one correct condition for the termination of the loop; A. alternative correct	
		logic for condition	
		8) WHILE loop, has correct 2 nd condition and correct logic, WHILE is	
		inside the selection structure; A. alternative correct logic for condition	
		9) Two correct assignment statements for x and L - inside the WHILE statement: P . If order of assignment statements incorrect	
		10) 2 nd WHILE loop, with syntax allowed by the programming language	
		and correct condition for the termination of the loop; R. If not inside 1 st	
		iterative structure	
		11) One correct assignment statement for F or X - inside the 2 nd WHILE	
		loop;	
		and the second sec	
		12) 2 ^{III} correct assignment statement for F or X - inside the 2 nd WHILE	
		loop; R. It order of assignment statements incorrect	
		13) Value of X outputted – must not be in an iterative structure and must	
		be in the THEN part of the selection structure;	
		14) Correct output monogon Matain and the	
		equal to 1 - must be in the ELSE part of the selection structure:	
		I. Case of variable names and output messages	
		A. IVIINOR TYPOS IN VARIABLE NAMES AND OUTPUT MESSAGES	
		 A initialisation of variables at declaration stage 	

5	20	****SCREEN CAPTURE**** Must match code from 19, including messages on screen capture	1
		matching those in code. Code for 19 must be sensible.	
		Mark as follows:	
		Value of 0.1 entered by the user followed by output message Not a	
		number greater than or equal to 1,	
5	21	****SCREEN CAPTURE****	1
		Must match code from 19, including messages on screen capture matching those in code. Code for 19 must be sensible.	
		Mark as follows: Value of 4.1 entered by the user followed by output of 2, 050025:	
		A. output of any number between 2 and 2.1	
5	22	(Estimate the) square root (of a number);	1
5	23	A (step-by-step) description of how to complete a task / a description of a	2
		process that achieves some task / a sequence of steps that solve a problem / a sequence of unambiguous instructions for solving a problem:	
		NE. Set of instructions	
		Independent of any programming language //	
		That can be completed in finite time;	
6	24	HumanPlayersTurn//	1
		MoveIsValid //	
		FlipStillPossible //	
		OpponentPieceFound;	
		A. Piece:	
		R. if any additional code	
		R. if spelt incorrectly	
6	25	DisplayGameBoard //	1
		CheckIfMoveIsValid //	
		GameOver //	
		PlayGame;	
		R. if any additional code	
		R. if spelt incorrectly	

0 10 RowCount // ColumnCount; 1 8 if any additional code R. if spelt incorrectly L. case 1 6 27 Count // Row// Column/Count // RowCount; 1 7 Row// Column/Count // RowCount; 1 8 R. if any additional code R. if spelt incorrectly L. case 1 6 28 Row // Column; 1 8 .if any additional code R. if spelt incorrectly L. case 1 6 28 Row // Column; 1 7 R. if any additional code R. if spelt incorrectly L. case 1 6 29 Choice // BoardSize // Move; 1 7 A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly L. case 1 7 30 ChangeBoardSize // GetLomputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetMenuthoice; 1 8 30 ChangeBoardSize // PrintLine // DisplayMenu // GetMenuthoice; 1 9 A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if any additional code 1	6	26	Saore //	1
RowCount // ColumnCount; R. if spel incorrectly 1 i. case 1 6 27 Count // Row // ColumnCount // Row // Column; 1 R. if any additional code R. if apel incorrectly I. case 6 29 Choice // BoardSize // Move; A. Coordinates A. playerName R. if apel incorrectly I. case 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetMenuChoice; A. Write	U	20		1
6 ColumnCount; R. if apy additional code 1 6 27 Count // Row // Columm // Columm // Column // R. if apy additional code R. if apelt incorrectly I. case 1 6 28 Row // Column; R. if any additional code R. if apelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayMenu // GetMenuChoice; 1 7 A. WriteNine (VB6 only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if apelt incorrectly 1			Rowcount //	
R. if any additional code R. if spelt incorrectly 1 6 27 Count // Row // ColumnCount // Row Count; 1 6 28 Row // Column; 1 6 28 Row // Column; 1 6 28 Row // Column; 1 7 R. if any additional code R. if spelt incorrectly I. case 1 6 28 Row // Column; 1 7 R. if any additional code R. if spelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 6 20 Choize // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetHumanPlayerMove // GetHumanPlayerMove // GetHumanPlayerMove // GetHumanPlayerMove // GetHumanPlayerMove // GetMenuChoice; 1 6 30 ChangeBoardSize // Movies 1 7 WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1			ColumnCount;	
R. if any additional code R. if spelt incorrectly 1 i. case 1 1 6 27 Count // Row // Column // Column // Column/count // RowCount; 1 R. if any additional code R. if any additional code 1 6 28 Row // Column; 1 7 R. if any additional code 1 8 Row // Column; 1 9 Choice // BoardSize // Move; 1 0 29 Choice // BoardSize // Move; 1 0 20 ChangeBoardSize // Move; 1 1 case 1 1 1 GetComputerPlayerName R. if any additional code R. if spelt incorrectly I. case 1 6 30 ChangeBoardSize // GetCamputerPlayerMove // GetCamputerPlayerMove // GetCamputerPlayerSname // PrintLine // DisplayGameBoard // R. if any additional code R. if spelt incorrectly 1				
R. if spelt incorrectly 1 6 27 Count // Row // Column // Column // Column // Column // Column // Column // Row Count; 1 7 Row // Row // Column // Column // Row // Column // Column // Row // Column; 1 6 28 Row // Column; 1 7 R. if any additional code R. if spelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerSName // PrintLine // DisplayGeneBoard // D			R. if any additional code	
I. case I. case 6 27 Count // Row // Column // R. if any additional code R. if spelt incorrectly I. case 1 6 28 Row // Column; R. if any additional code R. if spelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetMumanPlayerMove // Ge			R. if spelt incorrectly	
6 27 Count // Row// ColumnCount // RowCount; 1 7 Row // ColumnCount // RowCount; 1 8 R. if any additional code R. if spelt incorrectly I. case 1 6 28 Row // Column; R. if any additional code R. if spelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerNowe // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard			I. case	
6 27 Count // Row // Column // ColumnCount // RowCount; 1 R. if any additional code R. if spelt incorrectly 1 6 28 Row // Column; 1 6 28 Row // Column; 1 6 28 Row // Column; 1 6 29 Choice // Roorectly 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetPlayerSName // PrintLine // DisplayGameBoard (Python only) A. WriteLine (VB6 only) A. WriteLine (VB6 only) A. WriteLine (VB6 only) A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) R. if any additional code				
Row // Column // Column/out // RowCount;Row // Column/out // RowCount;R. if any additional code R. if spelt incorrectly I. case1628Row // Column; R. if any additional code R. if spelt incorrectly I. case1629Choice // BoardSize // Move;1629Choice // BoardSize // Move;1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetMenuChoice;1630ChangeBoardSize // GetMenuChoice;17MiteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only)	6	27	Count //	1
Image: Solution // Column // ColumnCount // RowCount; R. if any additional code R. if any additional code R. if spelt incorrectly Image: Column; 1 R. if any additional code 1 BoardSize // Move; 1 A. Coordinates 1 A. PlayerName 1 R. if any additional code R. if spelt incorrectly I. case 1 BoardSize // Move; 1 A. Coordinates 1 A. PlayerName 1 GetHumanPlayerMove // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // DisplayMenu // GetMenutChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. if spelt incorrectly R. if any additional code R. if spelt incorrectly	-		Pow //	-
6 28 Row // Column; 1 6 28 Row // Column; 1 6 28 Row // Column; 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayerSName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard (// DisplayGameDoard (// DisplayGameInchoice; 1 6 30 ChangeBoard (Python only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. Ki f any additional code R. if spet incorrectly			Column //	
6 28 Row // Column; 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameDard (// DisplayGameIode R, if spelt incorrectly 1 6 30 ChangeBoardSize // GetPlayersName // PrintLine // DisplayGameJoard // DisplayGameJoard // DisplayGameJoard // DisplayGameJoard (// DisplayGameJoard (// D				
RowCount;R. if any additional code R. if spelt incorrectly I. case1628Row // Column;1628Row // Column;1629Choice // BoardSize // Move;1629Choice // BoardSize // Move;1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // Printline // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard (// DisplayGameBoard (// DisplayGameBo			ColumnCount //	
R. if any additional code R. if spelt incorrectly I. case1628Row // Column; R. if any additional code R. if spelt incorrectly I. case1629Choice // BoardSize // Move; A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayerSname // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameDoard (// DisplayGameDoard (// D			RowCount;	
R. if any additional code R. if spelt incorrectly 1 6 28 Row // Column; 1 R. if any additional code R. if any additional code 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard (Python only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1				
R. if spelt incorrectly I. case 6 28 Row // Column; 1 R. if any additional code R. if any additional code 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 X. Coordinates 1 A. Coordinates 1 1 F. if any additional code R. if spelt incorrectly I. case 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly .			R. if any additional code	
I. case I. case 6 28 Row // Column; 1 R. if any additional code R. if spelt incorrectly 1 6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName (/ PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 6 XwriteLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1			R. if spelt incorrectly	
6 28 Row // Column; 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 1 6 29 Choice // BoardSize // Move; 1 A. Coordinates A. Coordinates 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. createBoard (Python only) R. if any additional code R. if spelt incorrectly			I. case	
6 28 Row // Column; 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 1 6 29 Choice // BoardSize // Move; 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 1 6 30 ChangeBoardSize // GetPlayerName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if spelt incorrectly				
0 10 Norm // Obtaining 1 R. if any additional code R. if spelt incorrectly I. case 1 6 29 Choice // BoardSize // Move; 1 A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1	6	28	Row // Column.	1
R. if any additional code R. if spelt incorrectly I. case1629Choice // BoardSize // Move;1A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayerSName // PrintLine // DisplayGameBoard // 	•			-
6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if any additional code 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetEngayerSName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1			P if any additional code	
6 29 Choice // 1 BoardSize // Move; 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 1 6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetHumanPlayerMove // 1 GetPlayersName // PrintLine // 1 JisplayGameBoard // DisplayGameBoard // 1 JisplayGameBoard // DisplayGameBoard // 1 A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly A. State and (Python only) A. State and (Python only)			R. if anot incorrectly	
6 29 Choice // BoardSize // Move; 1 6 29 Choice // BoardSize // Move; 1 A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1				
6 29 Choice // BoardSize // Move; 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if any additional code 1 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. reateBoard (Python only) Print any additional code R. if spelt incorrectly			I. Case	
6 29 Choice // BoardSize // Move; 1 BoardSize // Move; A. Coordinates 1 A. Coordinates A. PlayerName 1 R. if any additional code R. if spelt incorrectly 1 I. case 1 6 6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard (Python only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. FeadLine (VB6 only) A. if any additional code A. if any additional code A. if any additional code A				
BoardSize // Move; A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case I 6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetHumanPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly	6	29	Choice//	1
Move;Move;A. CoordinatesA. PlayerNameR. if any additional codeR. if spelt incorrectlyI. case630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice;A. WriteLine (VB6 only)A. WriteLine (VB6 only)A. ReadLine (VB6 only)A. CreateBoard (Python only)R. if any additional codeR. if spelt incorrectly			BoardSize //	
A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice;1A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly1			Move;	
A. Coordinates A. PlayerName R. if any additional code R. if spelt incorrectly I. case1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice;1A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly1				
A. PlayerName R. if any additional code R. if spelt incorrectly I. case1630ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // DisplayGameBoard // A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly1			A.Coordinates	
6 30 ChangeBoardSize // GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; 1 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 1			A. PlaverName	
6 30 ChangeBoardSize // 1 6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetComputerPlayerMove // 1 GetPlayersName // PrintLine // 1 DisplayGameBoard // DisplayGameBoard // 1 DisplayGameBoard // DisplayMenu // 1 GetMenuChoice; A. WriteLine (VB6 only) 1 A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) 1 A. CreateBoard (Python only) R. if any additional code 1			R if any additional code	
6 30 ChangeBoardSize // 1 6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetComputerPlayerMove // 1 GetPlayersName // PrintLine // 1 PrintLine // DisplayGameBoard // 1 DisplayGameBoard // DisplayMenu // 6 GetMenuChoice; A. WriteLine (VB6 only) 1 A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) 1 A. ReadLine (VB6 only) A. CreateBoard (Python only) 1 A. if any additional code R. if spelt incorrectly 1			P if spolt incorrectly	
6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly				
6 30 ChangeBoardSize // 1 GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly				
<pre>b 30 ChangeBoardSize// 1 GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly</pre>	_	00		4
GetHumanPlayerMove // GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly	6	30	ChangeBoardSize //	1
GetComputerPlayerMove // GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			GetHumanPlayerMove//	
GetPlayersName // PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			GetComputerPlayerMove//	
PrintLine // DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			GetPlayersName //	
DisplayGameBoard // DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			PrintLine //	
DisplayMenu // GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			DisplayGameBoard //	
GetMenuChoice; A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			DisplayMenu//	
A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly				
 A. WriteLine (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly 				
A. WriteBille (VB6 only) A. WriteNoLine (VB6 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			Λ Writeline ()/R6 only)	
A. WriteNoline (VB0 only) A. ReadLine (VB6 only) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			$\mathbf{A} = \frac{1}{2} \left(\frac{1}{2} \left(\frac{1}{2} \right) - \frac{1}{2} \left($	
A. ReadLine (VB6 ONIY) A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			A. WriteNoLine (VDO ONN)	
A. CreateBoard (Python only) R. if any additional code R. if spelt incorrectly			A. ReadLine (VB6 ONIY)	
R. if any additional code R. if spelt incorrectly			A. CreateBoard (Python only)	
R. if spelt incorrectly			R. if any additional code	
			R. if spelt incorrectly	
I. case			I. case	

```
6
    31
          VB.Net
                                                                               1
          Row = Move Mod 10 //
          MoveIsValid = False //
          MoveIsValid = True //
          Column = Move \setminus 10;
          Pascal
          Row := Move Mod 10 //
          Column := Move Div 10 //
          MoveIsValid := False //
          MoveIsValid := True //
          CheckIfMoveIsValid := MoveIsValid;
          I. semicolons
          VB6
          Row = Move Mod 10 //
          MoveIsValid = False //
          MoveIsValid = True //
          Column = Move \setminus 10 //
          CheckIfMoveIsValid = MoveIsValid;
          Java
          row = move % 10 //
          column = move / 10 //
          moveIsValid = false //
          moveIsValid = true;
          I. semicolons
          C#
          Row = Move % 10 //
          Column = Move / 10 //
          MoveIsValid = false //
          MoveIsValid = true;
          I. semicolons
          Python
          Row = Move % 10 //
          Column = Move // 10 /\!/
          MoveIsValid = False //
          MoveIsValid = True;
          R. if any additional code
          R. if spelt incorrectly
          I. case
```

6	22	Missing parameter (Decard) in the colle to	1
0	32	FlipOpperent Diagog In Ope Direction:	4
		ripopponencerecesinoneniteccion,	
		When FlipOpponentPiecesInOneDirection is called with the last	
		two parameters being 0 and 0 this will not be for a diagonal;	
		There are only three calls to FlipOpponentPiecesInOneDirection, there	
		should be 4 // Not all diagonals have been checked;	
		The subroutine calls have been placed before the variable Row has been	
		given a value // The subroutine calls have been placed before the	
		been placed before the variables have been given values // The	
		subrouting calls have been placed after the if statement.	
		subrouine cans have been placed after the instatement,	
6	33	When row 10 is entered Row will be assigned a value of 0;	2
		3	
		When 10 / a 2-digit row is entered the value of Column is always going to	
		be greater than 10 // when 10 / a 2-digit row is entered the value of	
		Column will contain part of the row number;	
		//	
		When the user tries to place a piece in 1010 then the program will try to	
		insert a piece in column 101 row 0	
		NE row / column would be incorrect	
		Note for examiners	
		To get 2 marks the answer must include the inputs that would be used by	
		the program to produce incorrect outputs for row and column and the	
		incorrect values that would be assigned to both row and column from	
		these inputs.	
7	34	Selection structure with correct condition in correct place in program code	3
		e.g. by checking for empty string or checking length of string is 0;	
		Correct message That is not a valid name using default	
		name instead displayed – must be inside the selection structure.	
		Default name Human player assigned to the PlayerName variable -	
		must be inside the selection structure;	
7	35	****SCREEN CAPTURE****	2
		Must match code from 34. Code for 34 must be sensible.	
		No name entered followed by error message from eads for 24 being	
		displayed.	
		Message saving Human player enter the letter of your	
		chosen option displayed:	
		· · · · · · · · · · · · · · · · · · ·	

8	36	Additional parameter in subroutine call to CheckIfMoveIsValid subroutine; R. If parameters do not match routine interface for code from 37	1
8	37	 Additional parameter in routine interface for CheckIfMoveIsValid subroutine; Selection structure with two correct conditions and correct logic; Selection structure with all four correct conditions and correct logic; Check for square containing a space is only done within their attempt at 2 and/or 3 i.e. when their conditions have been met; Correct Boolean values returned by subroutine under all circumstances; R. if check for boundary values are incorrect 	5
		A. answers that obtain correct functionality by just modifying the existing selection structure rather than creating a new selection structure. The check that Board(Row, Column) contains a space must be the last condition to award the final two mark points for 37 and short circuit evaluation must be used to connect this condition to the others. If short circuit evaluation has not been used and evidence has been provided showing that the modified program works then examiners should refer the answer to their team leader for advice (in case this functionality could be obtained by altering compiler settings).	

8	38	****SCREEN CAPTURE****	2
-		Must match code from 36 and 37. Code for 36 and 37 must be sensible.	
		Coordinates of 74 followed by message saying their name followed by	
		enter the coordinates for the square that you want to put a	
		piece in:	
		Coordinates of 47 followed by message saying their name followed by	
		enter the coordinates for the square that you want to put a	
		piece in:;	
		R. if no/incorrect upper boundary check in code for 37	
		Coordinates of 10 followed by message saying their name followed by	
		enter the coordinates for the square that you want to put a	
		piece in:	
		Coordinates of 01 followed by message saying their name followed by	
		enter the coordinates for the square that you want to put a	
		piece in:	
		Coordinates of 66 followed by game board being displayed with a H in	
		the bottom-right corner;	
		R. if no/incorrect lower boundary check in code for 37.	
		R. both marks if when coordinates entered are out of bounds the program	
		code from 37 would execute the check for an invalid square being empty.	
0	20	code from 37 would execute the check for an invalid square being empty.	7
9	39	 1) Appropriate message displayed to the user; 	7
9	39	 Code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice: 	7
9	39	 Code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. 1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to 	7
9	39	 Code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code: 	7
9	39	 Code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. 1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 4) Correct code will be executed if the user chooses the four centre 	7
9	39	 Code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; Code for four corner square option works correctly for two corners for 	7
9	39	 code from 37 would execute the check for an invalid square being empty. Appropriate message displayed to the user; Code that allows the user to enter their choice; Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; Correct code will be executed if the user chooses the four centre square option; Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; Code for four corner square option works correctly for two corners for all board sizes; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. 1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 4) Correct code will be executed if the user chooses the four centre square option; 5) Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 6) Code for four corner square option works correctly for two corners for all board sizes; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. 1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 4) Correct code will be executed if the user chooses the four centre square option; 5) Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 6) Code for four corner square option works correctly for two corners for all board sizes; 7) Code for four corner square option works correctly for all four corners for all board sizes; 	7
9	39	 code from 37 would execute the check for an invalid square being empty. 1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 4) Correct code will be executed if the user chooses the four centre square option; 5) Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 6) Code for four corner square option works correctly for two corners for all board sizes; 7) Code for four corner square option works correctly for all four corners for all board sizes; 	7

9	40	****SCREEN CAPTURE**** Must match code from 39, including prompts on screen capture matching those in code. Code for 39 must be sensible.	1
		User input for four centre square option followed by board display shown below;	
		НС	
		СН	
9	41	****SCREEN CAPTURE****	1
		those in code. Code for 39 must be sensible.	
		User input for four corner square option followed by board display shown below:	
		Н С	
0	40		1
9	42	Must match code from 39, including prompts on screen capture matching	I
		those in code. Code for 39 must be sensible.	
		User input for four corner square option followed by board display shown below:	
		НСС	
40	40		10
	43	minor typos	10

	2) Two parameters of the correct data type in the routine interface for new subroutine;	
	 3) Code that will cause message Choosing one of the following squares will mean you flip some pieces: to be displayed; A. Code for message in PlayGame 	
	4) Code will look at every row on the board;R. if not for all board sizes	
	5) Code will look at every column on the board;R. if not for all board sizes	
	 Note for examiners The following apply to mark points 6, 7 and 8 R. if will not work correctly for a square on the edge of the board. R. if possible flips are checked for a non-empty square. R. if squares get "permanently" changed when checking for flips I. Out of bounds checks 	
	6) Code correctly checks for flips in one direction;	
	7) Code correctly checks for flips in a second direction;	
	8) Code correctly checks for flips in all four directions	
	9) Selection structure that checks if square will result in flips;	
	10) Coordinates of square displayed – must be inside selection structure that checks if square will result in flips; A. alternative code which works correctly	
	MAX 9 if code will display any incorrect squares MAX 9 if code will not display any correct square	
	Note for examiners Refer unusual answers to Team Leader	

10	44	Correct subroutine call to GetFlipSquares subroutine; R. if subroutine call would not work I. Case Subroutine call added in correct place in program code;	2
10	45	<pre>****SCREEN CAPTURE**** Must match code from 43 and 44, including prompts on screen capture matching those in code. Code for 43 and 44 must be sensible. Message Choosing one of the following squares will mean you flip some pieces: is displayed followed by the values 24, 35, 53, 42; I. order of values displayed</pre>	1

Pascal

Qu	Part	Marking Guidance	Marks
5	19	Program Question5;	14
		Var	
		N : Real;	
		X : Real;	
		L : Real;	
		F : Real;	
		Begin	
		Write('Enter a number: ');	
		Readln(N);	
		F := 16;	
		If N >= 1	
		Then	
		Begin	
		X := N;	
		While $(X * X - N > 1)$ And $(F - 1 > 1)$	
		Do	
		Begin	
		L := X;	
		X := X / F;	
		While $(X * X \le N)$	
		Do	
		Begin	
		$\mathbf{F} := \mathbf{F} - 0 \cdot 1;$	
		X := L / F'	
		Endi	
		End;	
		WILLEIN(A),	
		Ella Elao Mritoln(UNot a number greater than or equal to	
		1:):	
		Peadln:	
		Fnd	
7	34	Function GetPlayersName : String;	3
•	0.	Var	Ũ
		PlayerName : String;	
		Begin	
		Write('What is your name? ');	
		Readln(PlayerName);	
		If PlayerName = ''	
		Then	
		Begin	
		Writeln('That is not a valid name, using default	
		<pre>name instead');</pre>	
		<pre>PlayerName := 'Human player';</pre>	
		End;	
		GetPlayersName := PlayerName;	
		End;	

8	36	<pre> Else Move := GetComputerPlayerMove(BoardSize); MoveIsValid := CheckIfMoveIsValid(Board, Move, BoardSize); Until MoveIsValid;</pre>	1
8	37	<pre>Function CheckIfMoveIsValid(Board : TBoard; Move, BoardSize : Integer) : Boolean; Var Row : Integer; Column : Integer; MoveIsValid : Boolean; Begin Row := Move Mod 10; Column := Move Div 10; MoveIsValid := False; If (Row <= BoardSize) And (Row >= 1) And (Column <= BoardSize) And (Column >= 1) Then If Board[Row, Column] = ' ' Then MoveIsValid := True; CheckIfMoveIsValid := MoveIsValid; End; Alternative answer</pre>	5
		<pre>Function CheckIfMoveIsValid(Board : TBoard; Move, BoardSize : Integer) : Boolean; MoveIsValid := False; If (Row > BoardSize) Or (Row < 1) Or (Column > BoardSize) Or (Column < 1) Then MoveIsValid := False Else If Board[Row, Column] = ' ' Then MoveIsValid := True; </pre>	

_

```
9
    39
                                                                         7
          Procedure SetUpGameBoard(Var Board : TBoard; BoardSize :
          Integer);
            Var
              Row : Integer;
              Column : Integer;
              Choice : Char;
            Begin
              Write('Do you want to play with the starting pieces
          being on the four c(e)ntre squares');
              Write('or on the four c(o)rner squares?');
              Readln(Choice);
              If Choice = 'e'
                Then
                  For Row := 1 To BoardSize
                    Do
                      For Column := 1 To BoardSize
                        Do
                           If (Row = (BoardSize + 1) Div 2) And
          (Column = (BoardSize + 1) Div 2 + 1)
                             Or (Column = (BoardSize + 1) Div 2) And
          (Row = (BoardSize + 1) Div 2 + 1)
                             Then Board[Row, Column] := 'C'
                             Else
                               If (Row = (BoardSize + 1) Div 2 + 1)
          And (Column = (BoardSize + 1) Div 2 + 1)
                                Or (Column = (BoardSize + 1) Div 2)
          And (Row = (BoardSize + 1) Div 2)
                                 Then Board[Row, Column] := 'H'
                                Else Board[Row, Column] := ' '
                Else
                  For Row := 1 To BoardSize
                    Do
                      For Column := 1 To BoardSize
                        Do
                           If (Row = 1) And (Column = 1)
                             Or (Row = BoardSize) And (Column =
          BoardSize)
                             Then Board[Row, Column] := 'H'
                             Else
                               If (Row = 1) And (Column = BoardSize)
                                 Or (Row = BoardSize) And (Column = 1)
                                 Then Board[Row, Column] := 'C'
                                 Else Board[Row, Column] := ' ';
            End;
```

```
Alternative answer
Procedure SetUpGameBoard(Var Board : TBoard; BoardSize :
Integer);
  Var
    Row : Integer;
    Column : Integer;
    Choice : Char;
  Begin
    Write('Do you want to play with the starting pieces
being on the four c(e)ntre squares');
    Write('or on the four c(o)rner squares?');
    Readln(Choice);
    If Choice = 'e'
      Then
        For Row := 1 To BoardSize
          Do
            For Column := 1 To BoardSize
              Do
                If (Row = (BoardSize + 1) Div 2) And
(Column = (BoardSize + 1) Div 2 + 1)
                  Or (Column = (BoardSize + 1) Div 2) And
(Row = (BoardSize + 1) Div 2 + 1)
                  Then Board[Row, Column] := 'C'
                  Else
                    If (Row = (BoardSize + 1) Div 2 + 1)
And (Column = (BoardSize + 1) Div 2 + 1)
                      Or (Column = (BoardSize + 1) Div 2)
And (Row = (BoardSize + 1) Div 2)
                      Then Board[Row, Column] := 'H'
                      Else Board[Row, Column] := ' '
      Else
        Begin
          For Row := 1 To BoardSize
            Do
              For Column := 1 To BoardSize
                Do
                  Board[Row, Column] := ' ';
          Board[1, 1] := 'H';
          Board[BoardSize, 1] := 'C';
          Board[1, BoardSize] := 'C';
          Board[BoardSize, BoardSize] := 'H';
        End;
  End;
```

10	43	Procedure GetFlipSquares(Board : TBoard; BoardSize :	10
		Integer);	
		Var	
		Row, Column: Integer;	
		SquareWillCauseFlips : Boolean;	
		OldValue : Char;	
		Begin	
		Writeln('Choosing one of the following squares will	
		mean you flip some pieces: ');	
		For Row $:= 1$ To BoardSize	
		For Column := 1 To BoardSize	
		If Board[Row Column] = ' ' Then	
		Begin	
		OldValue :- Peard[Pew Column]:	
		Board[Bow Column] :- 'H':	
		SourceWillCourseling :- Falco:	
		SquarewillCauserilps - raiser	
		DeardCize Dev Column 0 1)	
		BoardSize, Row, Column, 0, 1)	
		Inen SquarewiiiCauseriips ·= Irue,	
		II CheckIIThereArePieceSToFlip(Board,	
		BoardSize, Row, Column, 0, -1)	
		Then SquareWillCauseFlips := True;	
		II CheckIIInereArePiecesToFlip(Board,	
		BoardSize, Row, Column, I, U)	
		Then SquarewillCauseFilps := True;	
		II CheckIIInereArePiecesToFlip(Board,	
		BoardSize, Row, Column, -1 , 0)	
		Then SquareWillCauseFlips := True;	
		II SquareWillCauseFlips	
		Then	
		Begin	
		Write(Column);	
		Writeln(Row);	
		End;	
		Board[Row, Column] := OldValue;	
		Endi	
10	11		2
10	44		2
		The multiplayer sturn	
		Then	
		Begin Cotelingeners(Doord Doordding):	
		Getrippquares(Board, BoardSize);	
		Move := GetMove(PlayerName);	
		EISE MOVE := GetComputerPlayerMove(BoardSlze);	
		<pre>Movelsvalid := CneckliMovelsvalid(Board, Move, Decoderation);</pre>	
		BOARGSIZE);	
		UNTIL MOVELSVALIQ;	

VB.Net

Qu	Part	Marking Guidance	Marks
5	19	Dim N As Single	14
		Dim X As Single	
		Dim L As Single	
		Dim F As Single	
		Console.Write("Enter a number: ")	
		N = Console.ReadLine	
		F = 16	
		If $N \ge 1$ Then	
		X = N	
		While X * X - N > 1 And F - 1 > 1	
		L = X	
		X = X / F	
		While X * X <= N	
		F = F - 0.1	
		X = L / F	
		End While	
		End While	
		Console.WriteLine(X)	
		Else	
		Console.WriteLine("Not a number greater than or equal to	
		1")	
		End If	
		Console.ReadLine()	
7	34	Function GetPlayersName() As String	3
	•	Dim PlayerName As String	C .
		Console.Write("What is your name? ")	
		PlayerName = Console.ReadLine	
		If PlayerName = "" Then	
		Console.WriteLine("That is not a valid name, using	
		default name instead")	
		PlayerName = "Human player"	
		End If	
		Return PlayerName	
		End Function	
8	36		1
		End If	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)	
		Loop Until MoveIsValid	

```
8
    37
                                                                         5
          Function CheckIfMoveIsValid(ByVal Board(,) As Char, ByVal
          Move As Integer, ByVal BoardSize As Integer) As Boolean
            Dim Row As Integer
            Dim Column As Integer
            Dim MoveIsValid As Boolean
            Row = Move Mod 10
            Column = Move \setminus 10
            MoveIsValid = False
            If Row > BoardSize Or Row < 1 Or Column > BoardSize Or
          Column < 1 Then
              MoveIsValid = False
            ElseIf Board(Row, Column) = " " Then
              MoveIsValid = True
            End If
            Return MoveIsValid
          End Function
          Alternative answer
          Function CheckIfMoveIsValid(ByVal Board(,) As Char, ByVal
          Move As Integer, ByVal BoardSize As Integer) As Boolean
              If Row <= BoardSize And Row >= 1 And Column <=
          BoardSize And Column >= 1 Then
                If Board(Row, Column) = " " Then
                  MoveIsValid = True
                End If
              End If
              . . .
```

```
9
     39
                                                                              7
           Sub SetUpGameBoard(ByVal Board(,) As Char, ByVal BoardSize
           As Integer)
             Dim Row As Integer
             Dim Column As Integer
             Dim Choice As Char
             Console.Write("Do you want to play with the starting
           pieces being on the four c(e)ntre squares or on the four
           c(o)rner squares?")
             Choice = Console.ReadLine
             If Choice = "e" Then
               For Row = 1 To BoardSize
                 For Column = 1 To BoardSize
                    If Row = (BoardSize + 1) \setminus 2 And Column =
           (BoardSize + 1) \setminus 2 + 1 Or Column = (BoardSize + 1) \setminus 2 And
           Row = (BoardSize + 1) \setminus 2 + 1 Then
                      Board(Row, Column) = "C"
                    ElseIf Row = (BoardSize + 1) \setminus 2 + 1 And Column =
           (BoardSize + 1) \setminus 2 + 1 \text{ Or Column} = (BoardSize + 1) \setminus 2 \text{ And}
           Row = (BoardSize + 1) \setminus 2 Then
                      Board(Row, Column) = "H"
                    Else
                      Board(Row, Column) = " "
                   End If
                 Next
               Next
             Else
               For Row = 1 To BoardSize
                 For Column = 1 To BoardSize
                    If Row = 1 And Column = 1 Or Row = BoardSize And
           Column = BoardSize Then
                      Board(Row, Column) = "H"
                    ElseIf Row = 1 And Column = BoardSize Or Row =
           BoardSize And Column = 1 Then
                      Board(Row, Column) = "C"
                    Else
                      Board(Row, Column) = " "
                    End If
                 Next
               Next
             End If
           End Sub
```

```
Sub SetUpGameBoard(ByVal Board(,) As Char, ByVal BoardSize
As Integer)
  Dim Row As Integer
  Dim Column As Integer
  Dim Choice As Char
  Console.Write("Do you want to play with the starting
pieces being on the four c(e)ntre squares or on the four
c(o)rner squares?")
  Choice = Console.ReadLine
  If Choice = "e" Then
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        If Row = (BoardSize + 1) \setminus 2 And Column =
(BoardSize + 1) \setminus 2 + 1 Or Column = (BoardSize + 1) \setminus 2 And
Row = (BoardSize + 1) \setminus 2 + 1 Then
           Board(Row, Column) = "C"
        ElseIf Row = (BoardSize + 1) \setminus 2 + 1 And Column =
(BoardSize + 1) \setminus 2 + 1 \text{ Or Column} = (BoardSize + 1) \setminus 2 \text{ And}
Row = (BoardSize + 1) \setminus 2 Then
          Board(Row, Column) = "H"
        Else
          Board(Row, Column) = " "
        End If
      Next
    Next
  Else
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        BoardSize(Row, Column) = " "
      Next
    Next
    Board(1,1) = "H"
    Board(BoardSize, BoardSize) = "H"
    Board(1, BoardSize) = "C"
    Board(BoardSize, 1) = "C"
  End If
End Sub
```

Alternative answer

10	43	Sub GetFlipSquares(ByVal Board(,) As Char, ByVal BoardSize	10
		As Integer)	
		Dim Row As Integer	
		Dim Column As Integer	
		Dim SquareWillCauseFlips As Boolean	
		Dim OldValue As Char	
		Console.WriteLine("Choosing one of the following	
		squares will mean you flip some pieces: ")	
		For $Row = 1$ To BoardSize	
		For Column = 1 To BoardSize	
		If Board(Row Column) = " " Then	
		OldValue = Board(Row, Column)	
		Poard (Pow Column) - ""	
		SquareWillCaugeEling - Falge	
		If CheckIfTherePropherer in (Deerd	
		II CHECKIIIHEREAREPIECESIOFIIP(BOARD,	
		BoardSize, Row, Column, U, I) Inen	
		SquarewillCauseFlips = True	
		Elseli	
		CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column,	
		0, -1) Then	
		SquareWillCauseFlips = True	
		ElseIf	
		CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column,	
		1, 0) Then	
		SquareWillCauseFlips = True	
		ElseIf	
		CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column,	
		-1, 0) Then	
		SquareWillCauseFlips = True	
		End If	
		If SquareWillCauseFlips Then	
		Console.Write(Column)	
		Console.WriteLine(Row)	
		End If	
		Board(Row, Column) = OldValue	
		End If	
		Next	
		Next	
		Console.WriteLine()	
		End Sub	
10	44		2
		Do	
		If HumanPlayersTurn Then	
		GetFlipSquares(Board, BoardSize)	
		Move = GetMove(PlayerName)	
		Else	
		Move = GetComputerPlayerMove(BoardSize)	
		End If	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)	
		Loop Until MoveIsValid	

VB6

Qu	Part	Marking Guidance	Marks
5	19	Private Sub Form_Load()	14
		Dim N As Single	
		Dim X As Single	
		Dim L As Single	
		Dim F As Single	
		N = ReadLine("Enter a number: ")	
		F = 16	
		If $N \ge 1$ Then	
		X = N	
		While X * X - N > 1 And F - 1 > 1	
		L = X	
		X = X / F	
		While X * X <= N	
		F = F - 0.1	
		X = L / F	
		Wend	
		Wend	
		WriteLine(X)	
		Else	
		WriteLine("Not a number greater than or equal to 1")	
		End If	
		End Sub	
		Alternative answers could use some of the following instead of WriteLine: Console.Text = Console.Text & WriteLineWithMsg WriteWithMsg Msgbox WriteNoLine	
7	24		2
1	34	Dim DisverName Ag String	3
		Dim FlayerName - Readline("What is your name? ")	
		If PlayerName = "" Then	
		WriteLine ("That is not a valid name, using default	
		name instead")	
		PlayerName = "Human player"	
		End If	
		GetPlayersName = PlayerName	
		End Function	
8	36	•••	1
		End If	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)	
		Loop Until MoveIsValid	
		•••	

```
8
    37
                                                                          5
          Private Function CheckIfMoveIsValid(ByRef Board() As
          String, ByVal Move As Integer, ByVal BoardSize As Integer)
          As Boolean
            Dim Row As Integer
            Dim Column As Integer
            Dim MoveIsValid As Boolean
            Row = Move Mod 10
            Column = Move \setminus 10
            MoveIsValid = False
            If Row > BoardSize Or Row < 1 Or Column > BoardSize Or
          Column < 1 Then
              MoveIsValid = False
            ElseIf Board(Row, Column) = " " Then
              MoveIsValid = True
            End If
            CheckIfMoveIsValid = MoveIsValid
          End Function
          Alternative answer
          Private Function CheckIfMoveIsValid(ByRef Board() As
          String, ByVal Move As Integer, ByVal BoardSize As Integer)
          As Boolean
             . . .
              If Row <= BoardSize And Row >= 1 And Column <=
          BoardSize And Column >= 1 Then
                If Board(Row, Column) = " " Then
                  MoveIsValid = True
                End If
              End If
               . . .
```

```
9
     39
                                                                             7
           Private Sub SetUpGameBoard(ByRef Board() As String, ByVal
           BoardSize As Integer)
             Dim Row As Integer
             Dim Column As Integer
             Dim Choice As String
             Choice = ReadLine("Do you want to play with the starting
           pieces being on the four c(e)ntre squares or on the four
           c(o)rner squares?")
             If Choice = "e" Then
               For Row = 1 To BoardSize
                 For Column = 1 To BoardSize
                   If Row = (BoardSize + 1) \setminus 2 And Column =
           (BoardSize + 1) \setminus 2 + 1 Or Column = (BoardSize + 1) \setminus 2 And
           Row = (BoardSize + 1) \setminus 2 + 1 Then
                     Board(Row, Column) = "C"
                   ElseIf Row = (BoardSize + 1) \setminus 2 + 1 And Column =
           (BoardSize + 1) \setminus 2 + 1 Or Column = (BoardSize + 1) \setminus 2 And
           Row = (BoardSize + 1) \setminus 2 Then
                     Board(Row, Column) = "H"
                   Else
                     Board(Row, Column) = " "
                   End If
                 Next
               Next
             Else
               For Row = 1 To BoardSize
                 For Column = 1 To BoardSize
                   If Row = 1 And Column = 1 Or Row = BoardSize And
           Column = BoardSize Then
                      Board(Row, Column) = "H"
                   ElseIf Row = 1 And Column = BoardSize Or Row =
           BoardSize And Column = 1 Then
                     Board(Row, Column) = "C"
                   Else
                     Board(Row, Column) = " "
                   End If
                 Next
               Next
             End If
           End Sub
```

```
Alternative answer
Private Sub SetUpGameBoard(ByRef Board() As String, ByVal
BoardSize As Integer)
  Dim Row As Integer
  Dim Column As Integer
  Dim Choice As String
  Choice = ReadLine("Do you want to play with the starting
pieces being on the four c(e)ntre squares or on the four
c(o)rner squares?")
  If Choice = "e" Then
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
         If Row = (BoardSize + 1) \setminus 2 And Column =
(BoardSize + 1) \setminus 2 + 1 \text{ Or Column} = (BoardSize + 1) \setminus 2 \text{ And}
Row = (BoardSize + 1) \setminus 2 + 1 Then
           Board(Row, Column) = "C"
         ElseIf Row = (BoardSize + 1) \setminus 2 + 1 And Column =
(BoardSize + 1) \setminus 2 + 1 \text{ Or Column} = (BoardSize + 1) \setminus 2 \text{ And}
Row = (BoardSize + 1) \setminus 2 Then
           Board(Row, Column) = "H"
         Else
           Board(Row, Column) = " "
         End If
      Next
    Next
  Else
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
         BoardSize(Row, Column) = " "
      Next
    Next
    Board(1,1) = "H"
    Board(BoardSize, BoardSize) = "H"
    Board(1, BoardSize) = "C"
    Board(BoardSize, 1) = "C"
  End If
End Sub
```

<pre>BoardSize As Integer Dim Row As Integer Dim Column As Integer Dim SquareWillCauseFlips As Boolean Dim OldValue As String WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next WriteLine ("")</pre>
<pre>Dim Row As Integer Dim Row As Integer Dim Column As Integer Dim SquareWillCauseFlips As Boolean Dim OldValue As String WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = " " SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next WriteLine ("")</pre>
<pre>Dim Column As Integer Dim SquareWillCauseFlips As Boolean Dim OldValue As String WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize For Column = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Deard(Row, Column) = OldValue End If Next Next Next Next Next</pre>
<pre>Dim SquareWillCauseFlips As Boolean Dim OldValue As String WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteNoLine (Column) WriteIne (Row) End If Board(Row, Column) = OldValue End If Next Next Next Next Next</pre>
<pre>Dim SquarewillCauserIps As Boltean Dim OldValue As String WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize For Column = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteIne (Row) End If Board(Row, Column) = OldValue End If Next Next Next WriteLine ("")</pre>
<pre>WriteLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize For Column = 1 To BoardSize If Board(Row, Column) = " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteLine (Column) WriteLine (Row) End If Next Next Next WriteLine ("")</pre>
<pre>writeLine ("Choosing one of the following squares will mean you flip some pieces: ") For Row = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteLine (Column) WriteLine (Row) End If Next Next WriteLine ("")</pre>
<pre>mean you filp some pieces: ") For Row = 1 To BoardSize For Column = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = " " Then GaureWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteLine (Column) WriteLine (Column) WriteLine (Row) End If Next Next WriteLine ("")</pre>
<pre>For Row = 1 To BoardSize For Column = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteLine (Row) End If Next WriteLine ("")</pre>
<pre>For Column = 1 To BoardSize If Board(Row, Column) = " " Then OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Wext WriteLine ("") </pre>
<pre>If Board(Row, Column) = " " Then</pre>
<pre>OldValue = Board(Row, Column) Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next WriteLine ("")</pre>
<pre>Board(Row, Column) = "H" SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")</pre>
<pre>SquareWillCauseFlips = False If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next WriteLine ("")</pre>
<pre>If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")</pre>
<pre>BoardSize, Row, Column, 0, 1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next Next WriteLine ("")</pre>
SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next Next
<pre>ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")</pre>
<pre>BoardSize, Row, Column, 0, -1) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")</pre>
SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
BoardSize, Row, Column, 1, 0) Then SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
SquareWillCauseFlips = True ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next Next WriteLine ("")
ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
BoardSize, Row, Column, -1, 0) Then SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
<pre>SquareWillCauseFlips = True End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")</pre>
End If If SquareWillCauseFlips Then WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
<pre>If SquareWillCauseFlips Then</pre>
WriteNoLine (Column) WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
WriteLine (Row) End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
End If Board(Row, Column) = OldValue End If Next Next WriteLine ("")
Board(Row, Column) = OldValue End If Next Next WriteLine ("")
End If Next Next WriteLine ("")
Next Next WriteLine ("")
Next WriteLine ("")
WriteLine ("")
End Sub
10 44 2
Do
If HumanPlayersTurn Then
Call GetFlipSquares(Board, BoardSize)
Move = GetHumanPlayerMove(PlayerName)
Else
Move = GetComputerPlayerMove(BoardSize)
End If
MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)
Loop Until MoveIsValid

Java

```
Qu
    Part
          Marking Guidance
                                                                          Marks
5
    19
          public static void main(String[] args) {
                                                                          14
            float n;
            float x;
            float 1;
            float f;
            console.print("Enter a number: ");
            n = Float.parseFloat(console.readLine());
            f = 16;
            if (n >= 1) {
              x = n;
              while ((x * x - n > 1) \&\& (f - 1 > 1)) {
                1 = x;
                x = (float)(x / f);
                while (x * x <= n) {
                  f = (float)(f - 0.1);
                  x = (float)(l / f);
                }
              }
              console.println(x);
            }
            else {
              console.println("Not a number greater than or equal to
          1");
            }
          }
          Alternative answer (not using AQAConsole2016) :
          public static void main(String[] args) {
            Scanner in = new Scanner(System.in);
            float n;
            float x;
            float 1;
            float f;
            System.out.print("Enter a number: ");
            n = Float.parseFloat(in.next());
            f = 16;
            if (n >= 1) {
              x = n;
              while ((x * x - n > 1) \& (f - 1 > 1))
                1 = x;
                x = (float)(x / f);
                while (x * x \le n) {
                  f = (float)(f - 0.1);
                  x = (float)(l / f);
                 }
```

```
System.out.println(x);
  }
  else {
    System.out.println("Not a number greater than or equal
to 1");
  }
}
Alternative answer (using double variables)
public static void main(String[] args) {
  double n;
  double x;
  double 1;
  double f;
  console.print("Enter a number: ");
  n = Double.parseDouble(console.readLine());
  f = 16;
  if (n >= 1) {
    x = n;
    while ((x * x - n > 1) \&\& (f - 1 > 1)) 
      1 = x;
      x = x / f;
      while (x * x \le n) {
        f = f - 0.1;
        x = 1 / f ;
      }
    }
    console.println(x);
  }
  else {
    console.println("Not a number greater than or equal to
1");
  }
}
N.B. For n = 4.1 this gives output 2.04999999999958
```

```
7
    34
                                                                          3
          String getPlayersName() {
            String playerName;
            console.print("What is your name? ");
            playerName = console.readLine();
            if (playerName.equals("")){
              console.println("That is not a valid name, using
          default name instead");
              playerName = "Human player";
            }
            return playerName;
           }
          Alternative answer :
          String getPlayersName() {
            String playerName;
            console.print("What is your name? ");
            playerName = console.readLine();
            if (playerName.isEmpty()) {
              console.println("That is not a valid name, using
          default name instead");
              playerName = "Human player";
            }
            return playerName;
           }
          Alternative answer :
          String getPlayersName() {
            String playerName;
            console.print("What is your name? ");
            playerName = console.readLine();
            if (playerName.length() == 0) {
              console.println("That is not a valid name, using
          default name instead");
              playerName = "Human player";
            }
            return playerName;
          }
8
    36
                                                                          1
          . . .
            if (humanPlayersTurn) {
              move = getHumanPlayerMove(playerName);
            } else {
              move = getComputerPlayerMove(boardSize);
             }
            moveIsValid = checkIfMoveIsValid(board, move, boardSize);
           } while (!moveIsValid);
           . . .
```

```
8
    37
                                                                         5
          boolean checkIfMoveIsValid(char[][] board, int move, int
          boardSize) {
            int row;
            int column;
            boolean moveIsValid;
            row = move % 10;
            column = move / 10;
            moveIsValid = false;
            if (row <= boardSize && row >= 1 && column <= boardSize
          && column >= 1) {
              if (board[row][column] == ' ') {
                moveIsValid = true;
              }
            }
            return moveIsValid;
          }
          Alternative answer :
          boolean checkIfMoveIsValid(char[][] board, int move, int
          boardSize) {
            int row;
            int column;
            boolean moveIsValid;
            row = move % 10;
            column = move / 10;
            moveIsValid = false;
            if (row > boardSize || row < 1 || column > boardSize ||
          column < 1) {
              moveIsValid = false;
            } else {
              if (board[row][column] == ' ') {
                moveIsValid = true;
              }
            }
            return moveIsValid;
          }
```

```
39
                                                                          7
9
          void setUpGameBoard(char[][] board, int boardSize) {
            char choice;
            console.print("Do you want to play with the starting
          pieces being on the four c(e)ntre squares");
            console.print(" or on the four c(o)rner squares? ");
            choice = console.readChar();
            if (choice == 'e') {
              for (int row = 1; row <= boardSize; row++) {</pre>
                 for (int column = 1; column <= boardSize; column++) {</pre>
                   if (row == (boardSize + 1) / 2 && column ==
          (boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
          row == (boardSize + 1) / 2 + 1) {
                    board[row][column] = 'C';
                   } else {
                     if (row == (boardSize + 1) / 2 + 1 && column ==
          (boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
          row == (boardSize + 1) / 2) {
                       board[row][column] = 'H';
                     } else {
                       board[row][column] = ' ';
                     }
                   }
                 }
            } else {
              for (int row = 1; row <= boardSize; row++) {</pre>
                for (int column = 1; column <= boardSize; column++) {</pre>
                   if (row == 1 && column == 1 || row == boardSize &&
          column == boardSize) {
                     board[row][column] = 'H';
                   } else {
                     if (row == 1 && column == boardSize || row ==
          boardSize && column == 1) {
                      board[row][column] = 'C';
                     } else {
                       board[row][column] = ' ';
                     }
                  }
                }
              }
            }
          }
```

```
Alternative answer :
void setUpGameBoard(char[][] board, int boardSize) {
  char choice;
  console.print("Do you want to play with the starting
pieces being on the four c(e)ntre squares");
  console.print(" or on the four c(o)rner squares? ");
  choice = console.readChar();
  if (choice == 'e') {
    for (int row = 1; row <= boardSize; row++) {</pre>
      for (int column = 1; column <= boardSize; column++) {</pre>
        if (row == (boardSize + 1) / 2 && column ==
(boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
row == (boardSize + 1) / 2 + 1) {
          board[row][column] = 'C';
        } else {
          if (row == (boardSize + 1) / 2 + 1 && column ==
(boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
row == (boardSize + 1) / 2) {
            board[row][column] = 'H';
          } else {
            board[row][column] = ' ';
          }
        }
      }
    }
  } else {
    for (int row = 1; row <= boardSize; row++) {</pre>
      for (int column = 1; column <= boardSize; column++) {</pre>
        board[row][column] = ' ';
      }
    }
    board[1][1] = 'H';
    board[boardSize][1] = 'C';
    board[1][boardSize] = 'C';
    board[boardSize][boardSize] = 'H';
  }
}
```

10	43	void getFlipSquares(char[][] board, int boardSize) {	10
		boolean squareWillCauseFlips;	
		char oldValue;	
		console println("Choosing one of the following squares	
		will mean you flip some pieces"):	
		for (int row = 1; row <= boardSize: row++) \int	
		for (int column = 1; column <= boardSize; column \downarrow) (
		if $(\text{heard}[\text{rew}][\text{column}] = 1/2)$	
		$\prod_{i=1}^{n} \left(\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} \sum_{i=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} \sum_{i=1}^{n} \sum_{i=1}^{n} \sum_{i=1}^{n} \sum_{i=$	
		oldvalue = board[row][column];	
		board[row][column] = 'H';	
		squareWillCauseFlips = false;	
		if (checkIfThereArePiecesToFlip(board, boardSize,	
		row, column, 0, 1)) {	
		squareWillCauseFlips = true;	
		}	
		if (checkIfThereArePiecesToFlip(board, boardSize,	
		row, column, 0, -1)) {	
		squareWillCauseFlips = true;	
		}	
		if (checkIfThereArePiecesToFlip(board, boardSize,	
		row, column, 1, 0)) {	
		squareWillCauseFlips = true;	
		}	
		, if (checkIfThereArePiecesToFlip(board, boardSize,	
		row column -1 0))	
		gquareWillCaugeFling - true:	
		l squarewrricauserrips - crue/	
		j if (gguaroWillCaugoEling) (
		ii (squarewiiicauseriips) {	
		console.print(column);	
		console.princin(row),	
		<pre>board[row][column] = oldvalue;</pre>	
		}	
		}	
		}	
		console.println();	
		}	
			-
10	44		2
		do {	
		if (humanPlayersTurn) {	
		getFlipSquares(board, boardSize);	
		<pre>move = getHumanPlayerMove(playerName);</pre>	
		} else {	
		<pre>move = getComputerPlayerMove(boardSize);</pre>	
		}	
		<pre>moveIsValid = checkIfMoveIsValid(board, move, boardSize);</pre>	
		<pre>} while (!moveIsValid);</pre>	

Qu	Part	Marking Guidance	Marks
5	19	double X, F, L, N;	14
		Console.WriteLine("Enter a number:");	
		$\mathbf{F} = 16.0$:	
		if (N >= 1.0)	
		{	
		X = N;	
		while (X * X - N > 1.0 && F - 1.0 > 1.0)	
		L = X;	
		X = X / F;	
		while (X * X <= N)	
		$\mathbf{F} = \mathbf{F} = 0 \cdot \mathbf{I} \mathbf{i}$ $\mathbf{X} = \mathbf{I} \mathbf{i} \mathbf{F} \mathbf{i}$	
		}	
		}	
		Console.WriteLine(X);	
		} else	
		{	
		Console.WriteLine("Not a number greater than or equal to	
		1");	
		} Console ReadLine():	
7	34	<pre>static string GetPlayersName()</pre>	3
		{ string DlavorNamo:	
		Console Write ("What is your name? ");	
		PlayerName = Console.ReadLine();	
		if (PlayerName == "")	
		{	
		default name instead");	
		PlayerName = "Human Player";	
		}	
		return PlayerName;	
		}	
8	36		1
	50	Move = GetComputerPlayerMove(BoardSize);	
		}	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize);	
		j if (!HumanPlaversTurn)	
		····	

```
static bool CheckIfMoveIsValid(char[,] Board, int Move, int
8
    37
                                                                          5
          BoardSize)
          {
            int Row;
            int Column;
            bool MoveIsValid;
            Row = Move % 10;
            Column = Move / 10;
            MoveIsValid = false;
            if (Row < 1 || Row > BoardSize || Column < 1 || Column >
          BoardSize)
            {
              MoveIsValid = false;
            }
            else if (Board[Row, Column] == ' ')
            {
              MoveIsValid = true;
            }
            return MoveIsValid;
          }
          Alternative answer
          static bool CheckIfMoveIsValid(char[,] Board, int Move, int
          BoardSize)
          {
            if (Row >= 1 && Row <= BoardSize && Column >= 1 && Column
          <= BoardSize)
            {
              if (Board[Row, Column] == ' ')
               {
                MoveIsValid = true;
               }
            }
             • • •
          }
```

```
39
          static void SetUpGameBoard(char[,] Board, int BoardSize)
9
                                                                           7
           ł
             char Choice;
            Console.Write("Do you want to play with the starting
          pieces being on the four c(e)ntre squares or the four
          c(o)rner squares?");
            Choice = Convert.ToChar(Console.ReadLine());
             if (Choice == 'e')
             {
               for (int Row = 1; Row <= BoardSize; Row++)</pre>
               ł
                 for (int Column = 1; Column <= BoardSize; Column++)</pre>
                 {
                   if (Row == (BoardSize + 1) / 2 && Column ==
           (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2 &&
          Row == (BoardSize + 1) / 2 + 1)
                   ł
                     Board[Row, Column] = 'C';
                   }
                   else if (Row == (BoardSize + 1) / 2 + 1 && Column
          == (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2
          && Row == (BoardSize + 1) / 2)
                   {
                     Board[Row, Column] = 'H';
                   }
                   else
                   ł
                     Board[Row, Column] = ' ';
               }
             }
             else
             {
               for (int Row = 1; Row <= BoardSize; Row++)</pre>
               ł
                 for (int Column = 1; Column <= BoardSize; Column++)</pre>
                 {
                   if (Row == 1 && Column == 1 || Row == BoardSize &&
          Column == BoardSize)
                   {
                     Board[Row, Column] = 'H';
                   }
                   else if (Row == 1 && Column == BoardSize || Row ==
          BoardSize && Column == 1)
                   Ł
                     Board[Row, Column] = 'C';
                   }
                   else
                     Board[Row, Column] = ' ';
                   }
                 }
              }
            }
          }
```

```
Alternative answer
```

```
static void SetUpGameBoard(char[,] Board, int BoardSize)
ł
  char Choice;
  Console.Write("Do you want to play with the starting
pieces being on the four c(e)ntre squares or the four
c(o)rner squares?");
  Choice = Convert.ToChar(Console.ReadLine());
  if (Choice == 'e')
  {
    for (int Row = 1; Row <= BoardSize; Row++)</pre>
    ł
      for (int Column = 1; Column <= BoardSize; Column++)</pre>
      ł
        if (Row == (BoardSize + 1) / 2 && Column ==
(BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2 &&
Row == (BoardSize + 1) / 2 + 1)
        ł
          Board[Row, Column] = 'C';
        }
        else if (Row == (BoardSize + 1) / 2 + 1 && Column
== (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2
&& Row == (BoardSize + 1) / 2)
        {
          Board[Row, Column] = 'H';
        }
        else
        {
          Board[Row, Column] = ' ';
      }
    }
  }
  else
  {
    for (int Row = 1; Row <= BoardSize; Row++)</pre>
      for (int Column = 1; Column <= BoardSize; Column++)</pre>
      {
        Board[Row, Column] = ' ';
      }
      Board[1, 1] = 'H';
      Board[BoardSize, BoardSize] = 'H';
      Board[1, BoardSize] = 'C';
      Board[BoardSize, 1] = 'C';
    }
  }
}
```

10	43	<pre>static void GetFlipSquares(char[,] Board, int BoardSize)</pre>	10
		{	
		bool SquareWillCauseFlips;	
		char OldValue;	
		Console.WriteLine("Choosing one of the following squares	
		will mean you flip some pieces: ");	
		for (int Row = 1; Row <= BoardSize; Row++)	
		{	
		<pre>for (int Column = 1; Column <= BoardSize; Column++) {</pre>	
		<pre>if (Board[Row, Column] == ' ') {</pre>	
		OldValue = Board[Row, Column];	
		Board[Row, Column] = 'H';	
		SquareWillCauseFlips = false;	
		if (CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 0, 1))	
		{	
		SquareWillCauseFlips = true;	
		/ else if (CheckIfThereArePiecesToFlin(Board	
		BoardSize Row Column $(1 - 1)$	
		SquareWillCauseFlips = true;	
		else if (CheckIfThereArePiecesToFlip(Board.	
		BoardSize, Row, Column, 1, 0))	
		{	
		SquareWillCauseFlips = true;	
		<pre></pre>	
		BoardSize, Row, Column, -1, 0))	
		SquareWillCauseFlips = true;	
) if (SquareWillCauseFlips)	
		(Concola Write (Column):	
		Console Write(Column);	
		Seard[Pow Column] - OldValue:	
		}	
		Console.WriteLine();	
		}	

10	44		2
		do	
		{	
		if (HumanPlayersTurn)	
		{	
		GetFlipSquares(Board, BoardSize);	
		Move = GetHumanPlayerMove(PlayerName);	
		}	
		else	
		{	
		Move = GetComputerPlayerMove(BoardSize);	
		}	
		MoveIsValid = CheckIfMoveIsValid(Board, Move);	
		<pre>} while (!MoveIsValid);</pre>	

Python 2

Qu	Part	Marking Guidance	Marks
5	19	<pre>N = float(raw_input("Enter a number:"))</pre>	14
		F = 16.0	
		if N >= 1.0:	
		X = N	
		while X * X - N > 1.0 and F - 1.0 > 1.0:	
		L = X	
		X = X / F	
		while X * X <= N:	
		F = F - 0.1	
		X = L / F	
		print X	
		else:	
		print "Not a number greater than or equal to 1"	
7	34	def GetPlayersName():	3
		<pre>PlayerName = raw_input("What is your name? ")</pre>	
		if PlayerName == "":	
		print "That is not a valid name, using default name	
		instead"	
		PlayerName = "Human player"	
		return PlayerName	
		Alternative answer	
		def GetPlayersName():	
		<pre>PlayerName = raw_input("What is your name? ")</pre>	
		if len(PlayerName) == 0:	
		print "That is not a valid name, using default name	
		instead"	
		PlayerName = "Human player"	
		return PlayerName	
8	36		1
		else:	
		Move = GetComputerPlayerMove(BoardSize)	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)	

```
8
    37
                                                                         5
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move / 10
            MoveIsValid = False
            if Row > BoardSize or Row < 1 or Column > BoardSize or
          Column < 1:
              MoveIsValid = False
            elif Board[Row][Column] == " ":
              MoveIsValid = True
            return MoveIsValid
          Alternative answer
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move / 10
            MoveIsValid = False
            if Row <= BoardSize and Row >= 1 and Column <= BoardSize
          and Column >= 1:
              if Board[Row][Column] == " ":
                MoveIsValid = True
            return MoveIsValid
          Alternative answer
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move / 10
            MoveIsValid = False
            if Row <= BoardSize and Row >= 1 and Column <= BoardSize
          and Column >= 1 and Board[Row][Column] == " ":
              MoveIsValid = True
            return MoveIsValid
```

```
9
    39
                                                                         7
          def SetUpGameBoard(Board, BoardSize):
            Choice = raw_input("Do you want the starting pieces being
          on the four c(e)ntre squares or on the four c(o)rner
          squares?")
            if Choice == "e":
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if (Row == (BoardSize + 1) / 2 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2 + 1):
                    Board[Row][Column] = "C"
                  elif (Row == (BoardSize + 1) / 2 + 1 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2):
                    Board[Row][Column] = "H"
                  else:
                    Board[Row][Column] = " "
            else:
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if Row == 1 and Column == 1 or Row == BoardSize and
          Column == BoardSize:
                    Board[Row][Column] = "H"
                  elif Row == 1 and Column == BoardSize or Row ==
          BoardSize and Column == 1:
                    Board[Row][Column] = "C"
                  else:
                    Board[Row][Column] = " "
          Alternative answer
          def SetUpGameBoard(Board, BoardSize):
            Choice = raw_input("Do you want the starting pieces being
          on the four c(e)ntre squares or on the four c(o)rner
          squares?")
            if Choice == "e":
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if (Row == (BoardSize + 1) / 2 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2 + 1):
                    Board[Row][Column] = "C"
                  elif (Row == (BoardSize + 1) / 2 + 1 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2):
                    Board[Row][Column] = "H"
                  else:
                    Board[Row][Column] = " "
```

		else:	
1		for Row in range(1. BoardSize ± 1).	
		for Column in range(1) $PoardGize + 1$.	
		Poard[Port][Column] = " "	
		Board[1][1] = "H"	
		Board[BoardSize][BoardSize] = "H"	
		Board[1][BoardSize] = "C"	
		Board[BoardSize][1] = "C"	
10	43	def GetFlipSquares(Board, BoardSize):	10
		print "Choosing one of the following squares will mean	
		you will flip some pieces: "	
		for Row in range(1, BoardSize + 1):	
		for Column in range(1, BoardSize + 1):	
		if Board[Row][Column] == " ":	
		OldValue = Board[Row][Column]	
		Board[Row][Column] = "H"	
		SquareWillCauseFlips = False	
		if CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 0, 1):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, $0, -1$):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 1, 0):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, -1 , 0):	
		SquareWillCauseFlips = True	
		if SquareWillCauseFlips:	
		print_str(Column) + str(Row)	
		Board[Row][Column] = OldValue	
		print	
		Princ	
10	44		2
		while not MoveIsValid:	
		if HumanPlayersTurn:	
		GetFlipSquares(Board, BoardSize)	
		Move = GetHumanPlayerMove(PlayerName)	
		else:	
1			

Python 3

Qu	Part	Marking Guidance	Marks
5	19	<pre>N = float(input("Enter a number:"))</pre>	14
		F = 16.0	
		if N >= 1.0:	
		X = N	
		while $X * X - N > 1.0$ and $F - 1.0 > 1.0$:	
		L = X	
		X = X / F	
		while X * X <= N:	
		$\mathbf{F} = \mathbf{F} - 0.1$	
		X = L / F'	
		print(X)	
		erse.	
		princ(Not a number greater than of equal to 1)	
7	34	def GetPlaversName():	3
	•	PlayerName = input("What is your name? ")	C .
		if PlayerName == "":	
		print("That is not a valid name, using default name	
		instead")	
		PlayerName = "Human player"	
		return PlayerName	
		Alternative answer	
		DistorName - rate input ("What is your name? ")	
		<pre>if len(PlayerName) == 0.</pre>	
		print("That is not a valid name, using default name	
		instead")	
		PlayerName = "Human player"	
		return PlayerName	
		*	
8	36		1
		else:	
		Move = GetComputerPlayerMove(BoardSize)	
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)	
		•••	

```
8
    37
                                                                         5
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move // 10
            MoveIsValid = False
            if Row > BoardSize or Row < 1 or Column > BoardSize or
          Column < 1:
              MoveIsValid = False
            elif Board[Row][Column] == " ":
              MoveIsValid = True
            return MoveIsValid
          Alternative answer
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move // 10
            MoveIsValid = False
            if Row <= BoardSize and Row >= 1 and Column <= BoardSize
          and Column >= 1:
              if Board[Row][Column] == " ":
                MoveIsValid = True
            return MoveIsValid
          Alternative answer
          def CheckIfMoveIsValid(Board, Move, BoardSize):
            Row = Move % 10
            Column = Move // 10
            MoveIsValid = False
            if Row <= BoardSize and Row >= 1 and Column <= BoardSize
          and Column >= 1 and Board[Row][Column] == " ":
              MoveIsValid = True
            return MoveIsValid
```

```
9
    39
                                                                         7
          def SetUpGameBoard(Board, BoardSize):
            Choice = input("Do you want the starting pieces being on
          the four c(e)ntre squares or on the four c(o)rner
          squares?")
            if Choice == "e":
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if (Row == (BoardSize + 1) / 2 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2 + 1):
                    Board[Row][Column] = "C"
                  elif (Row == (BoardSize + 1) / 2 + 1 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2):
                    Board[Row][Column] = "H"
                  else:
                    Board[Row][Column] = " "
            else:
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if Row == 1 and Column == 1 or Row == BoardSize and
          Column == BoardSize:
                    Board[Row][Column] = "H"
                  elif Row == 1 and Column == BoardSize or Row ==
          BoardSize and Column == 1:
                    Board[Row][Column] = "C"
                  else:
                    Board[Row][Column] = " "
          Alternative answer
          def SetUpGameBoard(Board, BoardSize):
            Choice = input("Do you want the starting pieces being on
          the four c(e)ntre squares or on the four c(o)rner
          squares?")
            if Choice == "e":
              for Row in range(1, BoardSize + 1):
                for Column in range(1, BoardSize + 1):
                  if (Row == (BoardSize + 1) / 2 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2 + 1):
                    Board[Row][Column] = "C"
                  elif (Row == (BoardSize + 1) / 2 + 1 and Column ==
          (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2
          and Row == (BoardSize + 1) / 2):
                    Board[Row][Column] = "H"
                  else:
                    Board[Row][Column] = " "
```

		<pre>else: for Row in range(1, BoardSize + 1): for Column in range(1, BoardSize + 1): Board[Row][Column] = " "</pre>	
		Board[1][1] = "H"	
		Board[BoardSize][BoardSize] = "H"	
		Board[1][BoardSize] = "C"	
		Board[BoardSize][1] = "C"	
10	43	def GetFlipSquares(Board, BoardSize):	10
		print("Choosing one of the following squares will mean	
		you will flip some pieces: ")	
		for Row in range(1, BoardSize + 1):	
		for Column in range(1, BoardSize + 1):	
		if Board[Row][Column] == " ":	
		OldValue = Board[Row][Column]	
		Board[Row][Column] = "H"	
		SquareWillCauseFlips = False	
		if CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 0, 1):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 0, -1):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, 1, 0):	
		SquareWillCauseFlips = True	
		elif CheckIfThereArePiecesToFlip(Board, BoardSize,	
		Row, Column, -1, 0):	
		SquareWillCauseFlips = True	
		if SquareWillCauseFlips:	
		print(str(Column) + str(Row))	
		Board[Row][Column] = Oldvalue	
		princ()	
10	44		2
		while not MoveIsValid:	
		if HumanPlayersTurn:	
		GetFlipSquares(Board, BoardSize)	
		Move = GetHumanPlayerMove(PlayerName)	
		else:	