

---

# AS COMPUTER SCIENCE 7516/1

Paper 1

---

Mark scheme

June 2022

---

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 Examiner.

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

Further copies of this mark scheme are available from [aqa.org.uk](http://aqa.org.uk)

#### **Copyright information**

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.

Copyright © 2022 AQA and its licensors. All rights reserved.

## AS Computer Science

### Paper 1 (7516/1) – applicable to all programming languages A, B, C, D and E

June 2022

The following annotation is used in the mark scheme:

- ;** - means a single mark
- //** - means alternative response
- /** - means an alternative word or sub-phrase
- A.** - means acceptable creditworthy answer
- R.** - means reject answer as not creditworthy
- NE.** - means not enough
- I.** - means ignore
- DPT.** - means "Don't penalise twice". In some questions a specific error made by a candidate, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should only result in a candidate losing one mark, on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

## Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

### Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

### Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

Examiners are required to assign each of the candidate's responses to the most appropriate level according to **its overall quality**, and then allocate a single mark within the level. When deciding upon a mark in a level, examiners should bear in mind the relative weightings of the assessment objectives.

eg

In question **05**, the marks available are as follows:

AO1 (knowledge)	1 mark
AO2 (analyse)	1 mark

In question **13.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	2 marks
AO3 (programming)	6 marks

In question **14.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	3 marks
AO3 (programming)	9 marks

Where a candidate's answer only reflects one element of the AO, the maximum mark they can receive will be restricted accordingly.

Section A

Qu	Marks																																																																		
01	<div>5 marks for AO2 (apply)</div> <div><table><tr><th>S</th><th>X</th><th>Y</th><th>P</th><th>Z</th><th>List[Z]</th></tr><tr><td>38</td><td>0</td><td>18</td><td>-1</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>9</td><td>51</td></tr><tr><td></td><td></td><td>8</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>4</td><td>25</td></tr><tr><td></td><td>5</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>6</td><td>36</td></tr><tr><td></td><td>7</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>7</td><td>42</td></tr><tr><td></td><td></td><td>6</td><td></td><td></td><td></td></tr><tr><td colspan="5">Result:</td><td>-1</td></tr></table></div> <div>1 mark for each correct set of values in the correct sequence (boxed in red);</div> <div>A. entries within a boxed area sharing a row (e.g. 9 and 8 on the same row).</div> <div>Max 4 if any errors</div>	S	X	Y	P	Z	List[Z]	38	0	18	-1							9	51			8								4	25		5									6	36		7									7	42			6				Result:					-1
S	X	Y	P	Z	List[Z]																																																														
38	0	18	-1																																																																
				9	51																																																														
		8																																																																	
				4	25																																																														
	5																																																																		
				6	36																																																														
	7																																																																		
				7	42																																																														
		6																																																																	
Result:					-1																																																														

**Alternative Answer Layout:**

**1 mark** for each correct set of values in the correct sequence (4 sets boxed in red, 1 set in disjointed boxes in green and shaded light grey);

S	X	Y	P	Z	List[Z]
38	0	18	-1		
				9	51
		8		4	25
	5			6	36
	7			7	42
		6			
Result:					-1

Qu	Marks																										
02	<p><b>6 marks for AO2 (analyse)</b></p> <table border="1"> <thead> <tr> <th>Event / State</th><th>Label(s): (A) to (I), (W) to (Z)</th></tr> </thead> <tbody> <tr> <td>Card Payment Mode</td><td>X</td></tr> <tr> <td>Enter correct PIN</td><td>F</td></tr> <tr> <td>Enter incorrect PIN</td><td>D</td></tr> <tr> <td>Insert a coin (except final coin)</td><td>H</td></tr> <tr> <td>Insert final coin</td><td>I</td></tr> <tr> <td>Insert payment card</td><td>G</td></tr> <tr> <td>Paid Mode</td><td>Y</td></tr> <tr> <td>Payment Due Mode</td><td>Z</td></tr> <tr> <td>Press Accept</td><td>E</td></tr> <tr> <td>Press Cancel</td><td>C</td></tr> <tr> <td>Press + Button</td><td>A, B</td></tr> <tr> <td>Select Hours Mode</td><td>W</td></tr> </tbody> </table> <p><b>1 mark</b> per group of correct labels:  X, W  F, D  H, I, G  Y, Z  E, C    <b>A.</b> € ©  A, B</p> <p><b>I.</b> brackets around letters  <b>R.</b> any labels used more than once</p> <p><b>Max 5</b> if any errors</p>	Event / State	Label(s): (A) to (I), (W) to (Z)	Card Payment Mode	X	Enter correct PIN	F	Enter incorrect PIN	D	Insert a coin (except final coin)	H	Insert final coin	I	Insert payment card	G	Paid Mode	Y	Payment Due Mode	Z	Press Accept	E	Press Cancel	C	Press + Button	A, B	Select Hours Mode	W
Event / State	Label(s): (A) to (I), (W) to (Z)																										
Card Payment Mode	X																										
Enter correct PIN	F																										
Enter incorrect PIN	D																										
Insert a coin (except final coin)	H																										
Insert final coin	I																										
Insert payment card	G																										
Paid Mode	Y																										
Payment Due Mode	Z																										
Press Accept	E																										
Press Cancel	C																										
Press + Button	A, B																										
Select Hours Mode	W																										
	<b>6</b>																										



Qu		Marks
03	1	<p><b>8 marks for AO3 (programming)</b></p> <p><b>Mark as follows:</b></p> <p>1) Correct variable declarations for C, D, S, T and initialisation;</p> <p><b>Note to examiners:</b></p> <p>If a language allows variables to be used without explicit declaration, (eg Python), then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type.</p> <p>2) Correct <code>WHILE</code> loop syntax allowed by the programming language and correct condition for termination of the loop;</p> <p>3) Correct generation of two random numbers between 1 and 6 and output within loop;</p> <p>4) Correct running total assigned to S and updating of T;</p> <p>5) Correct condition to increment C within the loop;</p> <p>6) Correct condition to increment D with the loop;</p> <p>7) Correct calculation of A outside the loop;</p> <p>8) Correct output outside loop;</p> <p>I. case</p> <p><b>Max 7</b> if code does not function correctly</p>

03	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from <b>03.1</b>. Code for <b>03.1</b> must be sensible.</p> <p>As this output is from a random number generator, the output from candidates will not be the values below. However: Output should show two digits between 1 and 6 on each line except the final line. On the final line it should show the correct number of lines containing at least one 6, number of doubles and another integer. Number of lines containing at least one 6 and/or number of doubles must be 3</p> <p>Screen capture showing:</p> <pre>1 5 4 3 3 1 4 2 2 2 3 3 1 6 6 5 3 5 6 3 3 2 3 &gt;&gt;&gt;</pre> <p>I. missing spaces A. each digit on a new line</p>	1
----	---	--	---

## Section B

Qu		Marks	
04	1	<b>Mark is for AO1 (knowledge)</b>  Integer / int;	1
04	2	<b>Mark is for AO2 (analyse)</b>  Line / Row / Column / ErrorCount / ResponseNumber;  <b>A.</b> GRID_SIZE (Python only) <b>R.</b> Digit <b>R.</b> if any additional code <b>R.</b> if spelt incorrectly <b>I.</b> case and spacing	1
05		<b>1 mark for AO1 (knowledge) and 1 mark for AO2 (analyse)</b>  Identifier (AO2): OK / InputError / Finished / Solved / Correct / Incomplete; Data type (AO1): Boolean / bool;  <b>R.</b> if any additional code <b>R.</b> if spelt incorrectly <b>I.</b> case and spacing	2
06	1	<b>Mark is for AO2 (analyse)</b>  (Stores the string equivalent of User's) score;	1
06	2	<b>3 marks for AO2 (analyse)</b>  It stores (the string equivalent of) the number of digits/attempts/guesses/answers the player has placed/made // stores (the string equivalent of) the number of steps made (towards a solution);  <b>Max 1</b>  It is incremented every time a digit is accepted / valid; If the user has placed a digit // if the user has placed one or more digits: ... the (user's attempted) solution can be checked; ... the partial solution can be saved; ... the digit is placed into the (Answer) array at the correct position; ... the user's solution is displayed before exiting the program; ... the score is displayed;  <b>Max 2</b>	3

07		<p><b>2 marks for AO3 (design)</b></p> <p>Row/Column heading numbers need to be extended;  Row/Column headers formatted to occupy two positions;  Sub-grid lines need to be extended horizontally; <b>I.</b> vertically  Grid lines need to be extended horizontally; <b>I.</b> vertically  Sub-grid lines need to be after 4 rows / columns // column / row need to be divisible by 4 for sub-grid lines // the 3s need to be changed to 4s;</p> <p><b>R.</b> reference to alterations outside the <code>DisplayGrid</code> subroutine.</p> <p><b>Max 2</b></p>	2
08		<p><b>2 marks for AO1 (understand)</b></p> <p>Breaking down a/the problem into a number of sub-problems;  So that each sub-problem accomplishes an identifiable task;  Each of these sub-problems might be decomposed further;</p> <p><b>Max 2</b></p>	2
09	1	<p><b>2 marks for AO1 (understand)</b></p> <p>Definite iteration: the number of iterations is known (at the time of execution);  Indefinite iteration: the number of iterations depends on a condition (that is tested before / after each iteration);</p> <p><b>NE.</b> indefinite iteration: number of iterations not known</p>	2
09	2	<p><b>Mark is for AO2 (analyse)</b></p> <p><code>ResetDataStructures / LoadSolution / ResetAnswer /  TransferAnswerIntoGrid / DisplayGrid / KeepPuzzle /  CheckSolution / DisplayResults ;</code></p> <p><b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case and spacing</p>	1
09	3	<p><b>Mark is for AO2 (analyse)</b></p> <p><code>LoadPuzzleFile / TransferPuzzleIntoGrid  / LoadPartSolvedPuzzle / SolvePuzzle / GetMenuOption /  NumberPuzzle;</code></p> <p><b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case and spacing</p>	1

10	1	<p><b>2 marks for AO1 (understand)</b></p> <p>To stop a program from crashing;  To deal with (an anticipated) <u>run-time</u> error    <b>A.</b> clear description of an example of a run-time error;  To avoid structure of code becoming too complex as a result of trying to check for potential errors before they might occur;</p> <p><b>Max 2</b></p>	2
10	2	<p><b>2 marks for AO2 (analyse)</b>  <b>1 mark for identifier and 1 mark for exception</b></p> <p>LoadPuzzleFile – file does not exist // index error if more lines in file than Puzzle can accommodate;  LoadSolution – file does not exist // not enough lines in file;  TransferPuzzleIntoGrid – input conversion error // index error if string length &lt; 3;  LoadPartSolvedPuzzle – file does not exist // index error if more lines in file than Answer can accommodate;  SolvePuzzle – row/column/1<sup>st</sup> character/2<sup>nd</sup> character entered not a digit/number/integer // input conversion error;</p> <p><b>A.</b> KeepPuzzle – file does not exist (Java only)  <b>A.</b> readLine – end of file (Java only)</p> <p><b>A.</b> other exceptions that could occur</p> <p><b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case and spacing</p> <p><b>Max 2</b></p>	2
11	1	<p><b>Mark is for AO1 (knowledge)</b></p> <p>Named / (reference to) identifier / callable 'out of line' block of code (that may be called by simply writing its name in a program statement);</p> <p><b>A.</b> subroutine is used to give a block of code a name (and this name can be used to execute this group of statements)</p>	1
11	2	<p><b>Mark is for AO2 (analyse)</b></p> <p>LoadPuzzle is called more than once (from NumberPuzzle and also from LoadPartSolvedPuzzle)  // DisplayGrid is called more than once (from within SolvePuzzle)  // ResetDataStructures is called more than once (from NumberPuzzle and twice from LoadPuzzle);</p> <p><b>A.</b> readLine / print is called more than once (Java only)  <b>NE.</b> 'called more than once' without an appropriate subroutine name</p>	1

11	3	<p><b>Mark is for AO1 (understanding)</b></p> <p>Saves time as code can be written once and called in many places;  Reduced storage space of (source) code;  Reduces testing time as less code to test;  Avoids errors because code cannot be written differently if it is used in multiple places;  Program code will be easier to understand / maintain;</p> <p><b>MAX 1</b></p>	1
----	---	--	---

11	4	<b>2 marks for AO2 (analyse)</b>	2										
<table><tr><th>Box</th><th>Label</th></tr><tr><td>(a)</td><td>ResetDataStructures</td></tr><tr><td>(b)</td><td>LoadPuzzleFile</td></tr><tr><td>(c)</td><td>LoadSolution</td></tr><tr><td>(d)</td><td>TransferPuzzleIntoGrid</td></tr></table>				Box	Label	(a)	ResetDataStructures	(b)	LoadPuzzleFile	(c)	LoadSolution	(d)	TransferPuzzleIntoGrid
Box	Label												
(a)	ResetDataStructures												
(b)	LoadPuzzleFile												
(c)	LoadSolution												
(d)	TransferPuzzleIntoGrid												
<b>1 mark for 2 correct labels</b> <b>2 marks for 4 correct labels</b>													
<b>A.</b> labels written in different order <b>I.</b> case and spacing <b>R.</b> if any additional code <b>R.</b> if spelt incorrectly													

## Section C

Qu		Marks	
12	1	<p><b>7 marks for AO3 (programming)</b></p> <p><b>Marking guidance:</b></p> <p><b>Evidence of AO3 programming – 7 points:</b></p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> <li>1. Loop structure;</li> <li>2. Check through each element of <code>Puzzle</code> // check <code>Puzzle</code> until end of entries reached // check until a protected cell is found;</li> <li>3. Compare first two characters of <code>CellInfo</code> with first two characters of <code>Puzzle</code> line // compare the string values of <code>Row</code> and <code>Column</code> with the first two characters of the <code>Puzzle</code> line;</li> <li>4. If protected cell found ...;</li> <li>5. ... give appropriate error message to user <b>A.</b> 'Invalid input' <u>also</u> being output;</li> <li>6. Only if it is not a protected cell update content of cell; <b>DPT.</b> Incorrect identification of a protected cell</li> <li>7. Code in correct place in <code>SolvePuzzle</code>;</li> </ol> <p><b>Max 6</b> if any errors</p>	<b>7</b>

12	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from <b>12.1</b>, including prompts on screen capture matching those in code.</p> <p>Code for <b>12.1</b> must be sensible.</p> <p>Screen capture showing:</p> <pre> Enter your choice: S        1   2   3   4   5   6   7   8   9  ===.===.=== ===.===.=== ===.===.===  1  8 .   . 5     .   .       .   . 7    ..... ..... .....  2  9 .   .     5 . 7 . 4     .   .      ..... ..... .....  3  4 . 1 . 7     . 6 .       .   .      ===.===.=== ===.===.=== ===.===.===  4    .   .     7 .   .     1 . 6 .      ..... ..... .....  5  1 . 7 .     4 .   . 6     .   . 3    ..... ..... .....  6  6 . 5 . 8     .   . 1     .   .      ===.===.=== ===.===.=== ===.===.===  7    .   .       . 1 .       . 4 . 9    ..... ..... .....  8    .   .     2 .   . 7     .   . 1    ..... ..... .....  9  2 .   .       .   .     5 .   . 6    ===.===.=== ===.===.=== ===.===.===   Enter row column digit: (Press Enter to stop) 117 You can't change a protected cell Enter row column digit: (Press Enter to stop) 323 You can't change a protected cell Enter row column digit: (Press Enter to stop) 993 You can't change a protected cell Enter row column digit: (Press Enter to stop) 853 </pre>	1
----	---	--	---



```

      1   2   3   4   5   6   7   8   9
|===.===.===|===.===.===|===.===.===|
1| 8 . . 5 | . . . | . . 7 |
|.....|.....|.....|
2| 9 . . | 5 . 7 . 4 | . . |
|.....|.....|.....|
3| 4 . 1 . 7 | . 6 . | . . |
|===.===.===|===.===.===|===.===.===|
4| . . . | 7 . . | 1 . 6 . |
|.....|.....|.....|
5| 1 . 7 . | 4 . . 6 | . . 3 |
|.....|.....|.....|
6| 6 . 5 . 8 | . . 1 | . . |
|===.===.===|===.===.===|===.===.===|
7| . . . | . 1 . | . 4 . 9 |
|.....|.....|.....|
8| . . . | 2 . 3 . 7 | . . 1 |
|.....|.....|.....|
9| 2 . . | . . . | 5 . . 6 |
|===.===.===|===.===.===|===.===.===|

```

Enter row column digit:  
(Press Enter to stop)  
854

```

      1   2   3   4   5   6   7   8   9
|===.===.===|===.===.===|===.===.===|
1| 8 . . 5 | . . . | . . 7 |
|.....|.....|.....|
2| 9 . . | 5 . 7 . 4 | . . |
|.....|.....|.....|
3| 4 . 1 . 7 | . 6 . | . . |
|===.===.===|===.===.===|===.===.===|
4| . . . | 7 . . | 1 . 6 . |
|.....|.....|.....|
5| 1 . 7 . | 4 . . 6 | . . 3 |
|.....|.....|.....|
6| 6 . 5 . 8 | . . 1 | . . |
|===.===.===|===.===.===|===.===.===|
7| . . . | . 1 . | . 4 . 9 |
|.....|.....|.....|
8| . . . | 2 . 4 . 7 | . . 1 |
|.....|.....|.....|
9| 2 . . | . . . | 5 . . 6 |
|===.===.===|===.===.===|===.===.===|

```

**A.** grid displaying between incorrect attempts

Qu		Marks
13	1	<p><b>2 marks for AO3 (design) and 6 marks for AO3 (programming)</b></p> <p><b>Marking guidance:</b></p> <p><b>Evidence of AO3 design – 2 points:</b></p> <p>Evidence of design to look for in response:</p> <ol style="list-style-type: none"> <li>1. Identify the need for a loop containing a conditional statement;</li> <li>2. Recognise that subgrid boundaries need to be considered;</li> </ol> <p><b>Note: AO3</b> (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.</p> <p><b>Evidence of AO3 programming – 6 points:</b></p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> <li>3. Subroutine heading with correct parameters and return value;</li> <li>4. Check both row and column for duplicate <b>I.</b> failing to check location at which digit is to be placed;</li> <li>5. Correctly calculate all subgrid boundaries;</li> <li>6. Check each digit in any subgrid for duplicate (<i>does not depend on MP5 above</i>);</li> <li>7. Call in correct place in <code>SolvePuzzle</code> so that grid updated only under correct circumstances / if digit is valid / if subroutine returns true <b>R.</b> if returned value not used;</li> <li>8. Give appropriate error message (under at least some correct circumstances and is never displayed when it shouldn't be) <b>A.</b> error message in <code>DuplicateDigit</code>;</li> </ol> <p><b>Max 7</b> if any errors</p>

13	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from <b>13.1</b>, including prompts on screen capture matching those in code.</p> <p>Code for <b>13.1</b> must be sensible.</p> <p>Screen capture showing:</p> <pre> Enter your choice: S        1      2      3      4      5      6      7      8      9  ===.===.=== ===.===.=== ===.===.===  1  8 . . . 5   . . . .   . . . 7    ..... ..... .....  2  9 . . .   5 . . . 4   . . . .    ..... ..... .....  3  4 . 1 .   . 6 . .   . . . .    ===.===.=== ===.===.=== ===.===.===  4  . . . .   7 . . . .   1 . 6 . .    ..... ..... .....  5  1 . . .   4 . . . 6   . . . 3    ..... ..... .....  6  . 5 . 8   . . . 1   . . . .    ===.===.=== ===.===.=== ===.===.===  7  . . . .   . 1 . .   . 4 . 9    ..... ..... .....  8  . . . .   2 . . . 7   . . . 1    ..... ..... .....  9  2 . . .   . . . .   5 . . . 6    ===.===.=== ===.===.=== ===.===.===   Enter row column digit: (Press Enter to stop) 178 Duplicate digit Enter row column digit: (Press Enter to stop) 819 Duplicate digit Enter row column digit: (Press Enter to stop) 124 Duplicate digit Enter row column digit: (Press Enter to stop) 989 Duplicate digit Enter row column digit: (Press Enter to stop) 555 </pre>	1
----	---	--	---

	1	2	3	4	5	6	7	8	9
	===.===.===	===.===.===	===.===.===						
1	8 . . 5	. . .	. . .	. . .	. . .	. . .	. . .	. . .	7
	.....	.....	.....						
2	9 . .	5 . . 4	. . .	. . .	. . .	. . .	. . .	. . .	
	.....	.....	.....						
3	4 . 1 .	. . 6 .	. . .	. . .	. . .	. . .	. . .	. . .	
	===.===.===	===.===.===	===.===.===						
4	. . .	7 . .	1 . 6 .	. . .	. . .	. . .	. . .	. . .	
	.....	.....	.....						
5	1 . .	4 . 5 . 6	. . .	. . .	. . .	. . .	. . .	. . .	3
	.....	.....	.....						
6	. 5 . 8	. . . 1	. . .	. . .	. . .	. . .	. . .	. . .	
	===.===.===	===.===.===	===.===.===						
7	. . .	. 1 .	. 4 . 9	. . .	. . .	. . .	. . .	. . .	
	.....	.....	.....						
8	. . .	2 . . 7	. . . 1	. . .	. . .	. . .	. . .	. . .	
	.....	.....	.....						
9	2 . .	. . .	5 . . 6	. . .	. . .	. . .	. . .	. . .	
	===.===.===	===.===.===	===.===.===						

14

1

3 marks for AO3 (design) and 9 marks for AO3 (programming)

12

Level	Description	Mark Range
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.	9–12
2	There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work. This is a partially working programmed solution.	5–8
1	An attempt has been made to write the subroutine <code>ClearEntries</code> . Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1–4

**Marking guidance:**

**Evidence of AO3 design – 3 marks:**

Evidence of design to look for in response:

- 1) Recognise a loop required that repeats depending on value entered by user.
- 2) Dealing with non-integer input.
- 3) Attempt to identify minus sign / negative number in input.

**Note: AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.**

**Evidence of AO3 programming – 9 marks:**

Evidence of programming to look for in response:

- 4) Create `ClearEntries` subroutine with correct parameters.
- 5) Extract number of cells to be cleared.
- 6) Extract row/column from entry in `Answer` within loop.
- 7) Extract row and column from entry in `Answer` within loop.
- 8) Replace entry in `PuzzleGrid` with space within loop **DPT**. incorrect row and/or column
- 9) Correct number of cells cleared if number to clear is less than or equal to number of entries **DPT**. incorrect string used to clear in MP8
- 10) Correct number of cells cleared in all circumstances.
- 11) Update answer count in `Answer[2]`.
- 12) Display grid after subroutine call.

		<b>Max 11</b> if code does not function correctly	
<b>14</b>	<b>2</b>	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from <b>14.1</b>, including prompts on screen capture matching those in code. Code for <b>14.1</b> must be sensible.</p> <p>Enter your choice: S</p> <pre>       1      2      3      4      5      6      7      8      9  ===.===.=== ===.===.=== ===.===.===  1  8 . . . 5   . . . .   . . . 7    ..... ..... .....  2  9 . . .   5 . 7 . 4   . . .    ..... ..... .....  3  4 . 1 . 7   . 6 .   . . .    ===.===.=== ===.===.=== ===.===.===  4  . . .   7 . .   1 . 6 .    ..... ..... .....  5  1 . 7 .   4 . . 6   . . . 3    ..... ..... .....  6  6 . 5 . 8   . . . 1   . . .    ===.===.=== ===.===.=== ===.===.===  7  . . .   . 1 .   . 4 . 9    ..... ..... .....  8  . . .   2 . . 7   . . . 1    ..... ..... .....  9  2 . .   . . .   5 . . 6    ===.===.=== ===.===.=== ===.===.===  </pre> <p>Enter row column digit: (Press Enter to stop) -x not a valid integer Enter row column digit: (Press Enter to stop) -1</p>	<b>1</b>

	1	2	3	4	5	6	7	8	9
1	8	.	.	5	.	.	.	.	7
2	9	.	.	5	.	7	.	4	.
3	4	.	1	.	7	.	.	6	.
4	.	.	.	7	.	.	1	.	6
5	1	.	.	4	.	.	.	.	3
6	6	.	5	.	8	.	.	.	1
7	.	.	.	.	1	.	.	4	.
8	.	.	.	2	.	.	.	.	7
9	2	.	.	.	.	.	5	.	6

Enter row column digit:  
(Press Enter to stop)  
-5

	1	2	3	4	5	6	7	8	9
1	8	.	.	5	.	.	.	.	7
2	9	.	.	5	.	7	.	4	.
3	4	.	1	.	7	.	.	6	.
4	.	.	.	7	.	.	1	.	6
5	1	.	.	4	.	.	.	.	3
6	6	.	5	.	8	.	.	.	1
7	.	.	.	.	1	.	.	4	.
8	.	.	.	2	.	.	.	.	7
9	2	.	.	.	.	.	5	.	6

## VB.Net

03	1	<pre> Sub Main()     Dim Rnd As Random = New Random()     Dim C As Integer = 0 '     Dim D As Integer = 0 '     Dim S As Integer = 0 '     Dim T As Integer = 0 ' MP1     While C &lt; 3 And D &lt; 3 ' MP2         T += 1 '         Dim N1 As Integer = Rnd.Next(1, 7) '         Dim N2 As Integer = Rnd.Next(1, 7) ' MP3         Console.WriteLine(\$"{N1} {N2}") '         S = S + N1 + N2 ' MP4         If N1 = 6 Or N2 = 6 Then '             C += 1 ' MP5         End If         If N1 = N2 Then '             D += 1 ' MP6         End If     End While     Dim A As Integer = S \ (T * 2) ' MP7     Console.WriteLine(\$"{C} {D} {A}") ' MP8     Console.ReadLine() End Sub </pre>	8
12	1	<pre> Sub SolvePuzzle(PuzzleGrid(,) As Char, Puzzle() As String, Answer() As String)     Dim CellInfo As String     Dim InputError As Boolean     Dim Digit As Char     Dim Row, Column As Integer     DisplayGrid(PuzzleGrid)     If PuzzleGrid(0, 0) &lt;&gt; "X" Then         Console.WriteLine("No puzzle loaded")     Else         Console.WriteLine("Enter row column digit: ")         Console.WriteLine("(Press Enter to stop)")         CellInfo = Console.ReadLine()         While CellInfo &lt;&gt; EMPTY_STRING             InputError = False             If CellInfo.Length &lt;&gt; 3 Then                 InputError = True             Else                 Digit = CellInfo(2)                 Try                     Row = Int32.Parse(CellInfo(0))                 Catch                     InputError = True                 End Try                 Try                     Column = Int32.Parse(CellInfo(1))                 Catch                     InputError = True                 End Try                 If (Digit &lt; "1") Or (Digit &gt; "9") Then                     InputError = True                 End If             End If             If InputError Then                 Console.WriteLine("Invalid input")             Else </pre>	7



		<pre> ' MP7: code inserted in correct place Dim ProtectedCell As Boolean = False Dim Line As Integer = 0 While Not ProtectedCell And Puzzle(Line) &lt;&gt; EMPTY_STRING     ' MP1     ' MP2     If Puzzle(Line)(0) = Row.ToString() And Puzzle(Line)(1) = Column.ToString() Then ' MP3         ProtectedCell = True     End If     Line += 1 End While If ProtectedCell Then ' MP4     Console.WriteLine("You can't change a protected cell.") ' MP5 Else ' MP6     PuzzleGrid(Row, Column) = Digit     Answer(2) = (Int32.Parse(Answer(2)) + 1).ToString()     Answer(Int32.Parse(Answer(2)) + 2) = CellInfo     DisplayGrid(PuzzleGrid) End If End If Console.WriteLine("Enter row column digit: ") Console.WriteLine("(Press Enter to stop)") CellInfo = Console.ReadLine() End While End If End Sub </pre>	
13	1	<pre> Function DuplicateDigit(PuzzleGrid(,) As Char, Row As Integer, Column As Integer, Digit As Char) As Boolean ' MP3     Dim duplicateFound As Boolean = False     Dim i As Integer     For i = 1 To GRID_SIZE ' MP1         If PuzzleGrid(i, Column) = Digit Or PuzzleGrid(Row, i) = Digit Then             duplicateFound = True ' MP4         End If     Next     Dim endr As Integer = ((Row - 1) \ 3 + 1) * 3 ' MP2     Dim endc As Integer = ((Column - 1) \ 3 + 1) * 3     Dim r, c As Integer     For r = endr - 2 To endr         For c = endc - 2 To endc ' MP5             If PuzzleGrid(r, c) = Digit Then                 duplicateFound = True ' MP6             End If         Next     Next     Return duplicateFound End Function  Sub SolvePuzzle(PuzzleGrid(,) As Char, Puzzle() As String, Answer() As String)     Dim CellInfo As String     Dim InputError As Boolean     Dim Digit As Char     Dim Row, Column As Integer     DisplayGrid(PuzzleGrid)     If PuzzleGrid(0, 0) &lt;&gt; "X" Then         Console.WriteLine("No puzzle loaded")     Else         Console.WriteLine("Enter row column digit: ")         Console.WriteLine("(Press Enter to stop)")     End If End Sub </pre>	8

```

CellInfo = Console.ReadLine()
While CellInfo <> EMPTY_STRING
    InputError = False
    If CellInfo.Length <> 3 Then
        InputError = True
    Else
        Digit = CellInfo(2)
        Try
            Row = Int32.Parse(CellInfo(0))
        Catch
            InputError = True
        End Try
        Try
            Column = Int32.Parse(CellInfo(1))
        Catch
            InputError = True
        End Try
        If (Digit < "1") Or (Digit > "9") Then
            InputError = True
        End If
    End If
    If InputError Then
        Console.WriteLine("Invalid input")
    Else
        ' MP7: call subroutine in correct place
        If DuplicateDigit(PuzzleGrid, Row, Column, Digit) Then
            Console.WriteLine("Duplicate digit entered.") ' MP8
        Else
            PuzzleGrid(Row, Column) = Digit
            Answer(2) = (Int32.Parse(Answer(2)) + 1).ToString()
            Answer(Int32.Parse(Answer(2)) + 2) = CellInfo
            DisplayGrid(PuzzleGrid)
        End If
    End If
    Console.WriteLine("Enter row column digit: ")
    Console.WriteLine("(Press Enter to stop)")
    CellInfo = Console.ReadLine()
End While
End If
End Sub

Alternative Solution DuplicateDigit subroutine:

Function DuplicateDigit(PuzzleGrid(,) As Char, Row As Integer, Column As Integer,
Digit As Char) As Boolean ' MP3
    Dim duplicateFound As Boolean = False
    Dim i As Integer
    For i = 1 To GRID_SIZE ' MP1
        If PuzzleGrid(i, Column) = Digit Or PuzzleGrid(Row, i) = Digit Then
            duplicateFound = True ' MP4
        End If
    Next
    Dim startR As Integer = ((Row - 1) \ 3) * 3 + 1 ' MP2
    Dim startC As Integer = ((Column - 1) \ 3) * 3 + 1
    Dim r, c As Integer
    For r = startR To startR + 2
        For c = startC To startC + 2 ' MP5
            If PuzzleGrid(r, c) = Digit Then
                duplicateFound = True ' MP6
            End If
        Next
    Next
Next

```

		<pre> Return duplicateFound End Function </pre>	
14	1	<pre> Sub ClearEntries(PuzzleGrid(,) As Char, Answer() As String, EntriesToClear As Integer) 'MP4     Dim Line, Row, Column As Integer     Dim CellInfo As String     If EntriesToClear &gt; Int32.Parse(Answer(2)) Then 'MP9         EntriesToClear = Int32.Parse(Answer(2)) 'MP10     End If     For i As Integer = 0 To EntriesToClear - 1 'MP1         ' loop won't execute if Answer(2) = "0"         Line = Int32.Parse(Answer(2)) + 2         CellInfo = Answer(Line)         Row = Int32.Parse(CellInfo(0)) 'MP6         Column = Int32.Parse(CellInfo(1)) 'MP7         PuzzleGrid(Row, Column) = SPACE 'MP8         Answer(2) = (Int32.Parse(Answer(2)) - 1).ToString() 'MP11     Next End Sub  Sub SolvePuzzle(PuzzleGrid(,) As Char, Puzzle() As String, Answer() As String)     Dim CellInfo As String     Dim InputError As Boolean     Dim Digit As Char     Dim Row, Column As Integer     DisplayGrid(PuzzleGrid)     If PuzzleGrid(0, 0) &lt;&gt; "X" Then         Console.WriteLine("No puzzle loaded")     Else         Console.WriteLine("Enter row column digit: ")         Console.WriteLine("(Press Enter to stop)")         CellInfo = Console.ReadLine()         While CellInfo &lt;&gt; EMPTY_STRING             InputError = False             If CellInfo(0) = "-" Then 'MP3                 Try                     Dim EntriesToClear As Integer = Int32.Parse(CellInfo) * - 1 'MP5                     ClearEntries(PuzzleGrid, Answer, EntriesToClear)                     DisplayGrid(PuzzleGrid) 'MP12                 Catch ex As Exception 'MP2                     Console.WriteLine("Invalid number of entries to be cleared.")                 End Try             Else                 If CellInfo.Length &lt;&gt; 3 Then                     InputError = True                 Else                     Digit = CellInfo(2)                     Try                         Row = Int32.Parse(CellInfo(0))                     Catch                         InputError = True                     End Try                     Try                         Column = Int32.Parse(CellInfo(1))                     Catch                         InputError = True                     End Try                     If (Digit &lt; "1") Or (Digit &gt; "9") Then                         InputError = True                     End If                 End If             End If         End While     End If End Sub </pre>	12

```

End If
If InputError Then
    Console.WriteLine("Invalid input")
Else
    PuzzleGrid(Row, Column) = Digit
    Answer(2) = (Int32.Parse(Answer(2)) + 1).ToString()
    Answer(Int32.Parse(Answer(2)) + 2) = CellInfo
    DisplayGrid(PuzzleGrid)
End If
End If
End If
Console.WriteLine("Enter row column digit: ")
Console.WriteLine("(Press Enter to stop)")
CellInfo = Console.ReadLine()
End While
End If
End Sub

Alternative Solution
Sub ClearEntries(PuzzleGrid(,) As Char, Answer() As String, EntriesToClear As Integer)
    Dim Line, Row, Column As Integer
    Dim CellInfo As String
    Line = Int32.Parse (Answer(2)) + 2
    While EntriesToClear > 0 And Line > 2
        CellInfo = Answer(Line)
        Row = Int32.Parse(CellInfo(0))
        Column = Int32.Parse(CellInfo(1))
        PuzzleGrid(Row, Column) = SPACE
        Answer(2) = (Int32.Parse(Answer(2)) - 1).ToString()
        Line -= 1
        EntriesToClear -= 1
    End While
End Sub

```

## Python 3

03	1	<pre>import random  C = 0 # D = 0 # S = 0 # T = 0 # MP1 while C &lt; 3 and D &lt; 3: # MP2     T += 1     N1 = random.randint(1,6) #     N2 = random.randint(1,6) # MP3     print(N1, N2) #     S = S + N1 + N2 # MP4     if N1 == 6 or N2 == 6: #         C += 1 # MP5     if N1 == N2: #         D += 1 # MP6 A = S // (T*2) # MP7 print(C, D, A) # MP8</pre>	8
----	---	---	---

12	1	<pre> def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print("No puzzle loaded")     else:         print("Enter row column digit: ")         print("(Press Enter to stop)")         CellInfo = input()         while CellInfo != EMPTY_STRING:             InputError = False             if len(CellInfo) != 3:                 InputError = True             else:                 Digit = CellInfo[2]                 try:                     Row = int(CellInfo[0])                 except:                     InputError = True                 try:                     Column = int(CellInfo[1])                 except:                     InputError = True                 if (Digit &lt; '1' or Digit &gt; '9'):                     InputError = True             if InputError:                 print("Invalid input")             else:                 # MP7: code inserted in correct place                 Protected = False                 Line = 0                 while Puzzle[Line] != EMPTY_STRING:                     # MP1                     # MP2                     if CellInfo[:2] == Puzzle[Line][:2]: # MP3                         Protected = True                         Line += 1                 if Protected: # MP4                     print("You can't change a protected cell") # MP5                 else: # MP6                     PuzzleGrid[Row][Column] = Digit                     Answer[2] = str(int(Answer[2]) + 1)                     Answer[int(Answer[2]) + 2] = CellInfo                     DisplayGrid(PuzzleGrid)                 print("Enter row column digit: ")                 print("(Press Enter to stop)")                 CellInfo = input()         return PuzzleGrid, Answer </pre>	7
----	---	---	---

13	1	<pre> def DuplicateDigit(PuzzleGrid, Row, Column, Digit): # MP3     Duplicate = False     for X in range(1, GRID_SIZE + 1): # MP1         if PuzzleGrid[X][Column] == Digit: # MP4             Duplicate = True     for Y in range(1, GRID_SIZE + 1):         if PuzzleGrid[Row][Y] == Digit: # MP4             Duplicate = True     SubGridStartX = ((Row - 1) // 3) * 3 + 1 # MP2     SubGridStartY = ((Column - 1) // 3) * 3 + 1     for X in range(SubGridStartX, SubGridStartX + 3):         for Y in range(SubGridStartY, SubGridStartY + 3): # MP5             if PuzzleGrid[X][Y] == Digit:                 Duplicate = True # MP6     return Duplicate  def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print("No puzzle loaded")     else:         print("Enter row column digit: ")         print("(Press Enter to stop)")         CellInfo = input()         while CellInfo != EMPTY_STRING:             InputError = False             if len(CellInfo) != 3:                 InputError = True             else:                 Digit = CellInfo[2]                 try:                     Row = int(CellInfo[0])                 except:                     InputError = True                 try:                     Column = int(CellInfo[1])                 except:                     InputError = True             if (Digit &lt; '1' or Digit &gt; '9'):                 InputError = True             if InputError:                 print("Invalid input")             else:                 if DuplicateDigit(PuzzleGrid, Row, Column, Digit): # MP7                     print("Duplicate digit") # MP8                 else:                     PuzzleGrid[Row][Column] = Digit                     Answer[2] = str(int(Answer[2]) + 1)                     Answer[int(Answer[2]) + 2] = CellInfo                     DisplayGrid(PuzzleGrid)                     print("Enter row column digit: ")                     print("(Press Enter to stop)")                     CellInfo = input()         return PuzzleGrid, Answer </pre>	8
----	---	--	---

14	1	<pre> def ClearEntries(PuzzleGrid, Answer, NumberInput): # MP4     StepsDone = int(Answer[2])     if NumberInput &gt; StepsDone:         StepsToRemove = StepsDone # MP10     else:         StepsToRemove = NumberInput # MP9     for Count in range(StepsToRemove): # MP1         PreviousStep = Answer[int(Answer[2]) + 2]         Row = int(PreviousStep[0]) # MP6         Column = int(PreviousStep[1]) # MP7         PuzzleGrid[Row][Column] = SPACE # MP8         Answer[2] = str(int(Answer[2]) - 1) # MP11     return PuzzleGrid, Answer  def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print("No puzzle loaded")     else:         print("Enter row column digit: ")         print("(Press Enter to stop)")         CellInfo = input()         while CellInfo != EMPTY_STRING:             InputError = False             if CellInfo[0] == '-': # MP3                 try:                     NumberInput = -int(CellInfo) # MP5                     PuzzleGrid, Answer = ClearEntries(PuzzleGrid, Answer, NumberInput)                     DisplayGrid(PuzzleGrid) # MP12                 except: # MP2                     print("Not a valid integer")             else:                 if len(CellInfo) != 3:                     InputError = True                 else:                     Digit = CellInfo[2]                     try:                         Row = int(CellInfo[0])                         Column = int(CellInfo[1])                     except:                         InputError = True                     if (Digit &lt; '1' or Digit &gt; '9'):                         InputError = True                     if InputError:                         print("Invalid input")                     else:                         PuzzleGrid[Row][Column] = Digit                         Answer[2] = str(int(Answer[2]) + 1)                         Answer[int(Answer[2]) + 2] = CellInfo                         DisplayGrid(PuzzleGrid)                         print("Enter row column digit: ")                         print("(Press Enter to stop)")                         CellInfo = input()         return PuzzleGrid, Answer </pre>	12
----	---	--	----



	<p><b>Alternative solution</b></p> <pre> while CellInfo != EMPTY_STRING:     if CellInfo[0] == "-":         if CellInfo[1:].isdigit():             PuzzleGrid, Answer = ClearEntries(PuzzleGrid, Answer, CellInfo[1:])             DisplayGrid(PuzzleGrid)         else:             InputError = False  def ClearEntries(PuzzleGrid, Answer, Steps):     StepsToRemove = int(Steps)     for Count in range(StepsToRemove):         if int(Answer[2]) &gt; 0:             PreviousStep = Answer[int(Answer[2]) + 2]             Answer[int(Answer[2]) + 2] = EMPTY_STRING             Answer[2] = str(int(Answer[2]) - 1)             Row = int(PreviousStep[0])             Column = int(PreviousStep[1])             PuzzleGrid[Row][Column] = SPACE             #DisplayGrid(PuzzleGrid)     return PuzzleGrid, Answer </pre>	
--	--	--

## Python 2

03	1	<pre> import random  C = 0 # D = 0 # S = 0 # T = 0 # MP1 while C &lt; 3 and D &lt; 3: # MP2     T += 1 #     N1 = random.randint(1,6) #     N2 = random.randint(1,6) # MP3     print N1, N2 #     S = S + N1 + N2 # MP4     if N1 == 6 or N2 == 6: #         C += 1 # MP5     if N1 == N2: #         D += 1 # MP6 A = S // (T*2) # MP7 print C, D, A # MP8 </pre>	8
12	1	<pre> def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print "No puzzle loaded"     else:         print "Enter row column digit: "         print "(Press Enter to stop)"         CellInfo = raw_input()         while CellInfo != EMPTY_STRING:             InputError = False             if len(CellInfo) != 3:                 InputError = True             else:                 Digit = CellInfo[2]                 try:                     Row = int(CellInfo[0])                 except:                     InputError = True                 try:                     Column = int(CellInfo[1])                 except:                     InputError = True                 if (Digit &lt; '1' or Digit &gt; '9'):                     InputError = True             if InputError:                 print "Invalid input"             else:                 # MP7: code inserted in correct place                 Protected = False                 Line = 0                 while Puzzle[Line] != EMPTY_STRING:                     # MP1 MP2                     if CellInfo[:2] == Puzzle[Line][:2]: # MP3                         Protected = True                         Line += 1                 if Protected: # MP4 </pre>	7

		<pre>         print("You can't change a protected cell")      # MP5     else: # MP6         PuzzleGrid[Row][Column] = Digit         Answer[2] = str(int(Answer[2]) + 1)         Answer[int(Answer[2]) + 2] = CellInfo         DisplayGrid(PuzzleGrid)     print "Enter row column digit: "     print "(Press Enter to stop)"     CellInfo = raw_input()     return PuzzleGrid, Answer </pre>	
13	1	<pre> def DuplicateDigit(PuzzleGrid, Row, Column, Digit): # MP3     Duplicate = False     for X in range(1, GRID_SIZE + 1): # MP1         if PuzzleGrid[X][Column] == Digit: # MP4             Duplicate = True     for Y in range(1, GRID_SIZE + 1):         if PuzzleGrid[Row][Y] == Digit: # MP4             Duplicate = True     SubGridStartX = ((Row - 1) // 3) * 3 + 1 # MP2     SubGridStartY = ((Column - 1) // 3) * 3 + 1     for X in range(SubGridStartX, SubGridStartX + 3):         for Y in range(SubGridStartY, SubGridStartY + 3): # MP5             if PuzzleGrid[X][Y] == Digit:                 Duplicate = True # MP6     return Duplicate  def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print "No puzzle loaded"     else:         print "Enter row column digit: "         print "(Press Enter to stop)"         CellInfo = raw_input()         while CellInfo != EMPTY_STRING:             InputError = False             if len(CellInfo) != 3:                 InputError = True             else:                 Digit = CellInfo[2]                 try:                     Row = int(CellInfo[0])                 except:                     InputError = True                 try:                     Column = int(CellInfo[1])                 except:                     InputError = True             if (Digit &lt; '1' or Digit &gt; '9'):                 InputError = True             if InputError:                 print "Invalid input"             else:                 # MP7: call subroutine in correct place                 if DuplicateDigit(PuzzleGrid, Row, Column, Digit):                     print("Duplicate digit") # MP8 </pre>	8

		<pre> else:     PuzzleGrid[Row][Column] = Digit     Answer[2] = str(int(Answer[2]) + 1)     Answer[int(Answer[2]) + 2] = CellInfo     DisplayGrid(PuzzleGrid)     print "Enter row column digit: "     print "(Press Enter to stop)"     CellInfo = raw_input()     return PuzzleGrid, Answer </pre>	
14	1	<pre> def ClearEntries(PuzzleGrid, Answer, NumberInput): # MP4     StepsDone = int(Answer[2])     if NumberInput &gt; StepsDone:         StepsToRemove = StepsDone # MP10     else:         StepsToRemove = NumberInput # MP9     for Count in range(StepsToRemove): # MP1         PreviousStep = Answer[int(Answer[2]) + 2]         Row = int(PreviousStep[0]) # MP6         Column = int(PreviousStep[1]) # MP7         PuzzleGrid[Row][Column] = SPACE # MP8         Answer[2] = str(int(Answer[2]) - 1) # MP11     return PuzzleGrid, Answer  def SolvePuzzle(PuzzleGrid, Puzzle, Answer):     DisplayGrid(PuzzleGrid)     if PuzzleGrid[0][0] != 'X':         print "No puzzle loaded"     else:         print "Enter row column digit: "         print "(Press Enter to stop)"         CellInfo = raw_input()         while CellInfo != EMPTY_STRING:             InputError = False             if CellInfo[0] == '-': # MP3                 try:                     NumberInput = -int(CellInfo) # MP5                     PuzzleGrid, Answer = ClearEntries(PuzzleGrid, Answer, NumberInput)                     DisplayGrid(PuzzleGrid) # MP12                 except: # MP2                     print("Not a valid integer")             else:                 if len(CellInfo) != 3:                     InputError = True                 else:                     Digit = CellInfo[2]                     try:                         Row = int(CellInfo[0])                     except:                         InputError = True                     try:                         Column = int(CellInfo[1])                     except:                         InputError = True                     if (Digit &lt; '1' or Digit &gt; '9'):                         InputError = True </pre>	12

		<pre>         if InputError:             print "Invalid input"         else:             PuzzleGrid[Row][Column] = Digit             Answer[2] = str(int(Answer[2]) + 1)             Answer[int(Answer[2]) + 2] = CellInfo             DisplayGrid(PuzzleGrid)         print "Enter row column digit: "         print "(Press Enter to stop)"         CellInfo = raw_input()     return PuzzleGrid, Answer </pre>	
--	--	---	--

Pascal

03	1	<pre> {\$APPTYPE CONSOLE} {\$R+}  uses SysUtils; var A, C, D, S, T, N1, N2: integer; begin   C := 0; //   D := 0; //   S := 0; //   T := 0; // MP1   while (C &lt; 3) and (D &lt; 3) do // MP2   begin     T := T + 1; //     N1 := random(6) + 1; //     N2 := random(6) + 1; // MP3     writeln(N1, N2); //     S := S + N1 + N2; // MP4     if (N1 = 6) or (N2 = 6) then //       C := C + 1; // MP5     if N1 = N2 then //       D := D + 1; // MP6   end;   A := S div (T*2); // MP7   writeln(C, D, A); // MP8   readln; end. </pre>	8
12	1	<pre> procedure SolvePuzzle(var PuzzleGrid: TPuzzleGrid; Puzzle: TPuzzle; var Answer: TAnswer); var   CellInfo: string;   InputError: boolean;   Digit: char;   Line, Row, Column: integer;   Protected: boolean; begin   DisplayGrid(PuzzleGrid);   if PuzzleGrid[0, 0] &lt;&gt; 'X' then     writeln('No puzzle loaded')   else     begin       writeln('Enter row column digit: ');       writeln('(Press Enter to stop)');       readln(CellInfo);       while CellInfo &lt;&gt; EMPTY_STRING do         begin           InputError := False;           if length(CellInfo) &lt;&gt; 3 then             InputError := True           else             begin               Digit := CellInfo[3];               try                 Row := strToInt(CellInfo[1]);               except                 InputError := True;               end;               try </pre>	7

		<pre>         Column := strToInt(CellInfo[2]);     except         InputError := True;     end;     if (Digit &lt; '1') or (Digit &gt; '9') then         InputError := True;     end;     if InputError then         writeln('Invalid input')     else // MP7 code inserted in correct place     begin         Protected := False;         Line := 0;         while Puzzle[Line] &lt;&gt; EMPTY_STRING do // MP1, MP2         begin             if (CellInfo[1] = Puzzle[Line][1]) and (CellInfo[2] = Puzzle[Line][2]) then // MP3                 Protected := True;                 Inc(Line);             end;             if Protected then // MP4                 writeln('You can't change a protected cell') // MP5             else // MP6             begin                 PuzzleGrid[Row, Column] := Digit;                 Answer[2] := intToStr(strToInt(Answer[2]) + 1);                 Answer[strToInt(Answer[2]) + 2] := CellInfo;                 DisplayGrid(PuzzleGrid);             end;         end;         writeln('Enter row column digit: ');         writeln('(Press Enter to stop)');         readln(CellInfo);     end; end; end;</pre>	
13	1	<pre> function DuplicateDigit(PuzzleGrid: TPuzzleGrid; Row, Column: integer; Digit: string): boolean; // MP3 var     Duplicate: boolean;     X, Y, SubGridStartX, SubGridStartY: integer; begin     Duplicate := False;     for X := 1 to GRID_SIZE do // MP1         if PuzzleGrid[X, Column] = Digit then // MP4             Duplicate := True;     for Y := 1 to GRID_SIZE do         if PuzzleGrid[Row, Y] = Digit then // MP4             Duplicate := True;     SubGridStartX := ((Row - 1) div 3) * 3 + 1; // MP2     SubGridStartY := ((Column - 1) div 3) * 3 + 1;     for X := SubGridStartX to SubGridStartX + 2 do         for Y := SubGridStartY to (SubGridStartY + 2) do // MP5             if PuzzleGrid[X, Y] = Digit then                 Duplicate := True; // MP6     DuplicateDigit := Duplicate; end;</pre>	8

```

procedure SolvePuzzle(var PuzzleGrid: TPuzzleGrid; Puzzle:
TPuzzle; var Answer: TAnswer);
var
  CellInfo: string;
  InputError: boolean;
  Digit: char;
  Row, Column: integer;
begin
  DisplayGrid(PuzzleGrid);
  if PuzzleGrid[0, 0] <> 'X' then
    writeln('No puzzle loaded')
  else
    begin
      writeln('Enter row column digit: ');
      writeln('(Press Enter to stop)');
      readln(CellInfo);
      while CellInfo <> EMPTY_STRING do
        begin
          InputError := False;
          if length(CellInfo) <> 3 then
            InputError := True
          else
            begin
              Digit := CellInfo[3];
              try
                Row := strToInt(CellInfo[1]);
              except
                InputError := True;
              end;
              try
                Column := strToInt(CellInfo[2]);
              except
                InputError := True;
              end;
              if (Digit < '1') or (Digit > '9') then
                InputError := True;
            end;
          if InputError then
            writeln('Invalid input')
          else // MP7: call subroutine in correct place
            if DuplicateDigit(PuzzleGrid, Row, Column, Digit) then
              writeln('Duplicate digit') // MP8
            else
              begin
                PuzzleGrid[Row, Column] := Digit;
                Answer[2] := intToStr(strToInt(Answer[2]) + 1);
                Answer[strToInt(Answer[2]) + 2] := CellInfo;
                DisplayGrid(PuzzleGrid);
              end;
            writeln('Enter row column digit: ');
            writeln('(Press Enter to stop)');
            readln(CellInfo);
          end;
        end;
      end;
    end;
end;

```



14	1	<pre> <b>procedure</b> ClearEntries(<b>var</b> PuzzleGrid: TPuzzleGrid; <b>var</b> Answer: TAnswer; NumberInput: integer); // MP4 <b>var</b>     StepsDone, StepsToRemove, Count, Row, Column: integer;     PreviousStep: string; <b>begin</b>     StepsDone := strToInt(Answer[2]);     <b>if</b> NumberInput &gt; StepsDone <b>then</b>         StepsToRemove := StepsDone // MP10     <b>else</b>         StepsToRemove := NumberInput; // MP9     <b>for</b> Count := 1 <b>to</b> StepsToRemove <b>do</b> // MP1         <b>begin</b>             PreviousStep := Answer[strToInt(Answer[2]) + 2];             Row := strToInt(PreviousStep[1]); // MP6             Column := strToInt(PreviousStep[2]); // MP7             PuzzleGrid[Row, Column] := SPACE; // MP8             Answer[2] := intToStr(strToInt(Answer[2]) - 1); // MP11         <b>end</b>;     <b>end</b>;  <b>procedure</b> SolvePuzzle(<b>var</b> PuzzleGrid: TPuzzleGrid; Puzzle: TPuzzle; <b>var</b> Answer: TAnswer); <b>var</b>     CellInfo: string;     InputError: boolean;     Digit: char;     Row, Column, NumberInput: integer; <b>begin</b>     DisplayGrid(PuzzleGrid);     <b>if</b> PuzzleGrid[0, 0] &lt;&gt; 'X' <b>then</b>         writeln('No puzzle loaded')     <b>else</b>         <b>begin</b>             writeln('Enter row column digit: ');             writeln('(Press Enter to stop)');             readln(CellInfo);             <b>while</b> CellInfo &lt;&gt; EMPTY_STRING <b>do</b>                 <b>begin</b>                     InputError := False;                     <b>if</b> CellInfo[1] = '-' <b>then</b> // MP3                         <b>try</b>                             NumberInput := -strToInt(CellInfo); // MP5                             ClearEntries(PuzzleGrid, Answer, NumberInput);                             DisplayGrid(PuzzleGrid); // MP12                         <b>except</b> // MP2                             writeln('Not a valid integer')                         <b>end</b>                     <b>else</b>                         <b>begin</b>                             <b>if</b> length(CellInfo) &lt;&gt; 3 <b>then</b>                                 InputError := True                             <b>else</b>                                 <b>begin</b>                                     writeln('got past new code');                                     readln; </pre>	12
----	---	--	----

		<pre> Digit := CellInfo[3]; try     Row := strToInt(CellInfo[1]); except     InputError := True; end; try     Column := strToInt(CellInfo[2]); except     InputError := True; end; if (Digit &lt; '1') or (Digit &gt; '9') then     InputError := True; end; if InputError then     writeln('Invalid input') else begin     PuzzleGrid[Row, Column] := Digit;     Answer[2] := intToStr(strToInt(Answer[2]) + 1);     Answer[strToInt(Answer[2]) + 2] := CellInfo;     DisplayGrid(PuzzleGrid); <b>end;</b> end; writeln('Enter row column digit: '); writeln('(Press Enter to stop)'); readln(CellInfo); end; end; end;</pre>	
--	--	--	--

**C#**

03	1	<pre> Random R = new Random(); int C = 0, D = 0, S = 0, T = 0; // MP1 int N1, N2, A; while ((C &lt; 3) &amp;&amp; (D &lt; 3)) // MP2 {     T++; //     N1 = R.Next(1, 7); //     N2 = R.Next(1, 7); // MP3     Console.WriteLine(\$"{N1} {N2}"); //     S = S + N1 + N2; // MP4     if ((N1 == 6)    (N2 == 6)) //     { //         C++; // MP5     }     if (N1 == N2) //     { //         D++; // MP6     } } A = S / (T * 2); // MP7 Console.WriteLine(\$"{C} {D} {A}"); // MP8 Console.ReadLine(); </pre>	8
12	1	<pre> private static void SolvePuzzle(char[,] puzzleGrid, string[] puzzle, string[] answer) {     int row = 0, column = 0;     char digit = ' ';     DisplayGrid(puzzleGrid);     if (puzzleGrid[0, 0] != 'X')     {         Console.WriteLine("No puzzle loaded");     }     else     {         Console.WriteLine("Enter row column digit: ");         Console.WriteLine("(Press Enter to stop)");         string cellInfo = Console.ReadLine();         while (cellInfo != EMPTY_STRING)         {             bool inputError = false;             if (cellInfo.Length != 3)             {                 inputError = true;             }             else             {                 digit = cellInfo[2];                 try                 {                     row = Convert.ToInt32(cellInfo[0].ToString());                 }                 catch (Exception)                 {                     inputError = true;                 }             }         }     } } </pre>	7

		<pre>         }         try         {             column = Convert.ToInt32(cellInfo[1].ToString());         }         catch (Exception)         {             inputError = true;         }         if (digit &lt; '1'    digit &gt; '9')         {             inputError = true;         }     }     if (inputError)     {         Console.WriteLine("Invalid input");     }     else     { // MP7: code inserted in correct place         bool protectedCell = false;         int line = 0;         while (puzzle[line] != EMPTY_STRING) // MP1, MP2         {             if (cellInfo[0] == puzzle[line][0] &amp;&amp; cellInfo[1] == puzzle[line][1]) // MP3             {                 protectedCell = true;             }             line++;         }         if (protectedCell) // MP4         {             Console.WriteLine("You can't change a protected cell"); // MP5         }         else         { // MP6             puzzleGrid[row, column] = digit;             answer[2] = (Convert.ToInt32(answer[2]) + 1).ToString();             answer[Convert.ToInt32(answer[2]) + 2] = cellInfo;             DisplayGrid(puzzleGrid);         }         Console.WriteLine("Enter row column digit: ");         Console.WriteLine("(Press Enter to stop)");         cellInfo = Console.ReadLine();     } } } </pre>	
13	1	<pre> private static bool DuplicateDigit(char[,] puzzleGrid, int row, int column, char digit) // MP3 {     bool duplicate = false;     for (int x = 1; x &lt; GRID_SIZE + 1; x++) // MP1 </pre>	8

```

    {
        if (puzzleGrid[x,column] == digit) // MP4
        {
            duplicate = true;
        }
    }
    for (int y = 1; y < GRID_SIZE + 1; y++)
    {
        if (puzzleGrid[row,y] == digit) // MP4
        {
            duplicate = true;
        }
    }
    int subGridStartX = ((row - 1) / 3) * 3 + 1; // MP2
    int subGridStartY = ((column - 1) / 3) * 3 + 1;
    for (int x = subGridStartX; x < subGridStartX + 3; x++)
    {
        for (int y = subGridStartY; y < subGridStartY + 3; y++) //
MP5
        {
            if (puzzleGrid[x, y] == digit)
            {
                duplicate = true; // MP6
            }
        }
    }
    return duplicate;
}

private static void SolvePuzzle(char[,] puzzleGrid, string[]
puzzle, string[] answer)
{
    int row = 0, column = 0;
    char digit = ' ';
    DisplayGrid(puzzleGrid);
    if (puzzleGrid[0, 0] != 'X')
    {
        Console.WriteLine("No puzzle loaded");
    }
    else
    {
        Console.WriteLine("Enter row column digit: ");
        Console.WriteLine("(Press Enter to stop)");
        string cellInfo = Console.ReadLine();
        while (cellInfo != EMPTY_STRING)
        {
            bool inputError = false;
            if (cellInfo.Length != 3)
            {
                inputError = true;
            }
            else
            {
                digit = cellInfo[2];
                try

```

		<pre>         {             row = Convert.ToInt32(cellInfo[0].ToString());         }         catch (Exception)         {             inputError = true;         }         try         {             column = Convert.ToInt32(cellInfo[1].ToString());         }         catch (Exception)         {             inputError = true;         }         if (digit &lt; '1'    digit &gt; '9')         {             inputError = true;         }     }     if (inputError)     {         Console.WriteLine("Invalid input");     }     else     {         // MP7: call subroutine in correct place         if (DuplicateDigit(puzzleGrid, row, column, digit))         {             Console.WriteLine("Duplicate digit"); // MP8         }         else         {             puzzleGrid[row, column] = digit;             answer[2] = (Convert.ToInt32(answer[2]) + 1).ToString();             answer[Convert.ToInt32(answer[2]) + 2] = cellInfo;             DisplayGrid(puzzleGrid);         }     }     Console.WriteLine("Enter row column digit: ");     Console.WriteLine("(Press Enter to stop)");     cellInfo = Console.ReadLine(); } } } </pre>	
14	1	<pre> private static void ClearEntries(char[,] puzzleGrid, string[] answer, int numberInput) // MP4 {     int stepsToRemove = 0, row, column;     string previousStep = "";     int stepsDone = Convert.ToInt32(answer[2]);     if (numberInput &gt; stepsDone)     {         stepsToRemove = stepsDone; // MP10     }     else </pre>	12

```

{
    stepsToRemove = numberInput; // MP9
}
for (int count = 0; count < stepsToRemove; count++) // MP1
{
    previousStep = answer[Convert.ToInt32(answer[2]) + 2];
    row = Convert.ToInt32(previousStep[0].ToString()); // MP6
    column = Convert.ToInt32(previousStep[1].ToString()); // MP7
    puzzleGrid[row, column] = SPACE; // MP8
    answer[2] = (Convert.ToInt32(answer[2]) - 1).ToString(); //
MP11
}
}

private static void SolvePuzzle(char[,] puzzleGrid, string[]
puzzle, string[] answer)
{
    int row = 0, column = 0;
    char digit = ' ';
    int numberInput;
    DisplayGrid(puzzleGrid);
    if (puzzleGrid[0, 0] != 'X')
    {
        Console.WriteLine("No puzzle loaded");
    }
    else
    {
        Console.WriteLine("Enter row column digit: ");
        Console.WriteLine("(Press Enter to stop)");
        string cellInfo = Console.ReadLine();
        while (cellInfo != EMPTY_STRING)
        {
            bool inputError = false;
            if (cellInfo[0] == '-') // MP3
            {
                try
                {
                    numberInput = -Convert.ToInt32(cellInfo); // MP5
                    ClearEntries(puzzleGrid, answer, numberInput);
                    DisplayGrid(puzzleGrid); // MP12
                }
                catch (Exception) // MP2
                {
                    Console.WriteLine("Not a valid integer");
                }
            }
            else
            {
                if (cellInfo.Length != 3)
                {
                    inputError = true;
                }
                else
                {
                    digit = cellInfo[2];
                    try

```

```
{
    row = Convert.ToInt32(cellInfo[0].ToString());
}
catch (Exception)
{
    inputError = true;
}
try
{
    column = Convert.ToInt32(cellInfo[1].ToString());
}
catch (Exception)
{
    inputError = true;
}
if (digit < '1' || digit > '9')
{
    inputError = true;
}
}
if (inputError)
{
    Console.WriteLine("Invalid input");
}
else
{
    puzzleGrid[row, column] = digit;
    answer[2] = (Convert.ToInt32(answer[2]) +
1).ToString();
    answer[Convert.ToInt32(answer[2]) + 2] = cellInfo;
    DisplayGrid(puzzleGrid);
}
}
Console.WriteLine("Enter row column digit: ");
Console.WriteLine("(Press Enter to stop)");
cellInfo = Console.ReadLine();
}
}
```



## Java

03	1	<pre> Random r = new Random(); int c = 0; // int d = 0; // int s = 0; // int t = 0; // MP1 int n1, n2, a; while ((c &lt; 3) &amp;&amp; (d &lt; 3)) // MP2 {     t++; //     n1 = r.nextInt(6) + 1; //     n2 = r.nextInt(6) + 1; // MP3     Console.WriteLine(n1 + " " + n2); //     s = s + n1 + n2; // MP4     if ((n1 == 6)    (n2 == 6)) //     { //         c++; // MP5     }     if (n1 == n2) //     { //         d++; // MP6     } } a = s / (t * 2); // MP7 Console.WriteLine(c + " " + d + " " + a); // MP8 </pre>	8
12	1	<pre> void solvePuzzle(char[][] puzzleGrid, String[] puzzle, String[] answer) {     String cellInfo;     boolean inputError;     char digit = ' ';     int row = 0, column = 0;     displayGrid(puzzleGrid);     if (puzzleGrid[0][0] != 'X')     {         Console.WriteLine("No puzzle loaded");     }     else     {         Console.WriteLine("Enter row column digit: ");         Console.WriteLine("(Press Enter to stop)");         cellInfo = Console.ReadLine();         while (!cellInfo.equals(EMPTY_STRING))         {             inputError = false;             if (cellInfo.length() != 3)             {                 inputError = true;             }             else             {                 digit = cellInfo.charAt(2);                 try                 { </pre>	7

```

        row = Integer.parseInt(cellInfo.charAt(0) +
EMPTY_STRING);
    }
    catch (Exception ex)
    {
        inputError = true;
    }
    try
    {
        column = Integer.parseInt(cellInfo.charAt(1)
+ EMPTY_STRING);
    }
    catch (Exception ex)
    {
        inputError = true;
    }
    if ((digit < '1') || (digit > '9'))
    {
        inputError = true;
    }
}
if (inputError)
{
    Console.WriteLine("Invalid input");
}
else
{
    // MP7: code inserted in correct place
    boolean protectedValue = false;
    int line = 0;
    while (!puzzle[line].equals(EMPTY_STRING)) { //
MP1, MP2
        if (cellInfo.charAt(0) ==
puzzle[line].charAt(0) &&
            cellInfo.charAt(1) ==
puzzle[line].charAt(1)) { // MP3
            protectedValue = true;
        }
        line++;
    }
    if (protectedValue) { // MP4
        Console.WriteLine("You can't change a
protected cell");// MP5
    }
    else
    { // MP6
        puzzleGrid[row][column] = digit;
        answer[2] = (Integer.parseInt(answer[2]) +
1) + EMPTY_STRING;
        answer[Integer.parseInt(answer[2]) + 2] =
cellInfo;
        displayGrid(puzzleGrid);
    }
}
Console.WriteLine("Enter row column digit: ");
Console.WriteLine("(Press Enter to stop)");
cellInfo = Console.ReadLine();

```

		<pre>         }     } } </pre>	
13	1	<pre> boolean duplicateDigit(char [][] puzzleGrid, int row, int column, char digit) // MP3 {     boolean duplicate = false;     for (int x = 1; x &lt;= GRID_SIZE; x++) { // MP1         if (puzzleGrid[x][column] == digit) { // MP4             duplicate = true;         }     }     for (int y = 1; y &lt;= GRID_SIZE; y++) {         if (puzzleGrid[row][y] == digit) { // MP4             duplicate = true;         }     }     int subGridStartX = ((row - 1) / 3) * 3 + 1; // MP2     int subGridStartY = ((column - 1) / 3) * 3 + 1;     for (int x = subGridStartX; x &lt; subGridStartX + 3; x++) {         for (int y = subGridStartY; y &lt; subGridStartY + 3; y++)         { // MP5             if (puzzleGrid[x][y] == digit) {                 duplicate = true; // MP6             }         }     }     return duplicate; }  void solvePuzzle(char [][] puzzleGrid, String[] puzzle, String[] answer) {     String cellInfo;     boolean inputError;     char digit = ' ';     int row = 0, column = 0;     displayGrid(puzzleGrid);     if (puzzleGrid[0][0] != 'X')     {         Console.WriteLine("No puzzle loaded");     }     else     {         Console.WriteLine("Enter row column digit: ");         Console.WriteLine("(Press Enter to stop)");         cellInfo = Console.ReadLine();         while (!cellInfo.equals(EMPTY_STRING))         {             inputError = false;             if (cellInfo.length() != 3)             {                 inputError = true;             }             else             { </pre>	8

		<pre> digit = cellInfo.charAt(2); try {     row = Integer.parseInt(cellInfo.charAt(0) + EMPTY_STRING); } catch (Exception ex) {     inputError = true; } try {     column = Integer.parseInt(cellInfo.charAt(1) + EMPTY_STRING); } catch (Exception ex) {     inputError = true; } if ((digit &lt; '1')    (digit &gt; '9')) {     inputError = true; } } if (inputError) {     Console.WriteLine("Invalid input"); } else {     // MP7: call subroutine in correct place     if (duplicateDigit(puzzleGrid, row, column, digit)) {         Console.WriteLine("Duplicate digit");// MP8     }     else     {         ... </pre>	
--	--	---	--

14	1	<pre> void clearEntries(char [][] puzzleGrid, String[] answer, int numberInput) // MP 4 {     int stepsToRemove = 0;     int stepsDone = Integer.parseInt(answer[2]);     if (numberInput &gt; stepsDone) {         stepsToRemove = stepsDone; // MP 10     }     else     {         stepsToRemove = numberInput; // MP 9     }     for (int count = 0; count &lt; stepsToRemove; count++) { // MP 1         String previousStep = answer[Integer.parseInt(answer[2])+2];         int row = Integer.parseInt(previousStep.charAt(0) + EMPTY_STRING); // MP 6         int column = Integer.parseInt(previousStep.charAt(1) + EMPTY_STRING); // MP 7         puzzleGrid[row][column] = SPACE; // MP 8         answer[2] = (Integer.parseInt(answer[2]) - 1) + EMPTY_STRING; // MP 11     } }  void solvePuzzle(char[][] puzzleGrid, String[] puzzle, String[] answer) {     String cellInfo;     boolean inputError;     char digit = ' ';     int row = 0, column = 0, numberInput;     displayGrid(puzzleGrid);     if (puzzleGrid[0][0] != 'X')     {         Console.WriteLine("No puzzle loaded");     }     else     {         Console.WriteLine("Enter row column digit: ");         Console.WriteLine("(Press Enter to stop)");         cellInfo = Console.readLine();         while (!cellInfo.equals(EMPTY_STRING))         {             inputError = false;             if (cellInfo.charAt(0) == '-') { // MP 3                 try {                     numberInput = - Integer.parseInt(cellInfo); // MP 5                     clearEntries(puzzleGrid, answer, numberInput);                      displayGrid(puzzleGrid); // MP 12                 } catch (Exception e) { // MP 2                     Console.WriteLine("Not a valid integer");                 }             }         }     } } </pre>	12
----	---	---	----

		<pre>         }     else     {         if (cellInfo.length() != 3)         {             inputError = true;         }         else         {             digit = cellInfo.charAt(2);         }     } } </pre>	
		...	