1		**All marks for AO1 (understanding)** | Identifier	Description
---	---
`PlainTextLetter`	**uncoded letter, part of** `PlainText`
`Signal`	**single unit of** `Transmission` (`=` **or** `SPACE` **or** `EOL`)
`FirstSignal`	**first character in** `Transmission`
`Symbol`	**used to build** `SymbolString` (`.` **or** `-`)	 **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing **1 mark** per correct identifier **Note:** each identifier must only be used once (if given more than once, ignore all occurrences)	4

2	1	**All marks for AO2 (analyse)** the variable is used as the index of /pointer to / iterator for / place value; … the current character/symbol/signal in the transmission string;	2
2	2	**All marks for AO2 (analyse)** 1) empty string returned from `StripLeadingSpaces` (and assigned to `Transmission)` // generate empty string in `StripLeadingSpaces`; 2) `StripLeadingSpaces` (calls ReportError) to display "No signal received" 3) empty string is returned to `ReceiveMorseCode` // empty string is returned from `GetTransmission`; 4) `LastChar` is set to -1; 5) so loop is not entered; 6) `MorseCodeString` / `PlainText` remain empty strings; **Max 5**	5
3		**All marks for AO2 (analyse)** 1) Any example string with only two consecutive symbols, for example "==" // any example string with more than 3 consecutive symbols, such as "====" ; ***Note:*** *"xxx" would not cause an error.* ***Note:*** *It does not matter which non-space symbol is used in transmission.* 2) The while loop counts the number of consecutive non-spaces; 3) If this number is not 0, 1 or 3, (it calls the `ReportError` subroutine); **I**. quotes	3

4		**All marks AO2 (analyse)** **mark as follows:** 1) include digits 0 to 9 in `Letter` array; 2) include Morse codes in `MorseCode` array for digit characters; 3) extend `Dash` and `Dot` array; 4) at the corresponding positions some of the zeros will need to change (to include new pointers) // binary tree to include routes to digit characters; 5) `Decode` subroutine needs no changes; 6) `SendMorseCode` needs to test for digits; 7) Explain a method to look up Morse code for digits (eg linear search of `Letter` array); Unusual answers should be referred to the PE **A**. answers using dictionary for digits **A**. Letter and MorseCode arrays need changing; for **1 mark** if (1) and (2) not awarded **Max 6**	6

5		**Mark is for AO1 (understand)**
		
		`MoveRecord`
		`/ListOfMoves;`
		
		R. if any additional code
		R. if spelt incorrectly
		I. case & spacing
			1

6		**Mark is for AO2 (analyse)** positions of player A's pieces; **A** the contents of (the data structure/variable) `A` // pointer/address to `A` // `A`;	**1**

7	**1**	**All marks for AO2 (analyse)** (row 0 column 0) is used to store the number of moves; (row 0 column 1) is used to store the number of pieces promoted to dames;	**2**
7	**2**	**Mark for AO2 (analyse)** There are (a maximum of) 12 pieces per player // each row stores data for each piece;	**1**
7	**3**	**All marks for AO2 (analyse)** rows 1 to 12 (in columns 0 and 1) store the coordinates/location of the pieces on the board; if coordinates are -1 then indicates no piece; (column 2) indicates if the piece is a dame // indicates state of each piece; **Max 2**	**2**
8		**All marks for AO2 (analyse)** it counts the number of moves that are possible at the current state of play; it acts as the index for the data structure `ListOfMoves`;	**2**

<table>
<tr><td>9</td><td>1</td><td>Mark is for AO2 (analyse)

<code>OpponentsPieces</code>;

R. if any additional code
R. if spelt incorrectly
I. case & spacing</td><td>1</td></tr>
<tr><td>9</td><td>2</td><td>2 marks for AO3 (design) and 7 marks for AO3 (programming)

<table>
<tr><th>Level</th><th>Description</th><th>Mark Range</th></tr>
<tr><td>3</td><td>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.</td><td>7–9</td></tr>
<tr><td>2</td><td>There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.</td><td>4–6</td></tr>
<tr><td>1</td><td>An attempt has been made to amend the subroutine <code>MoveDame</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.</td><td>1–3</td></tr>
</table>

Marking guidance:

Evidence of AO3 design – 2 points:

Evidence of design to look for in response:

1) validate that chosen piece is an opponent’s existing piece
2) return updated <code>OpponentsPieces</code> from subroutine <code>MoveDame</code> (parameter by reference)

Evidence of AO3 programming – 7 points:

Evidence of programming to look for in response:

3) user prompt for which piece to take
4) extracting player letter from chosen piece
5) extracting index from chosen piece
6) retrieving coodinates from <code>OpponentsPieces</code>
7) set opponent’s piece coordinates to -1
8) new dame’s coordinates set to taken piece’s coordinates
9) update parameters in calls to <code>MovePiece</code> in subroutine <code>MakeMove</code> (parameter by reference)

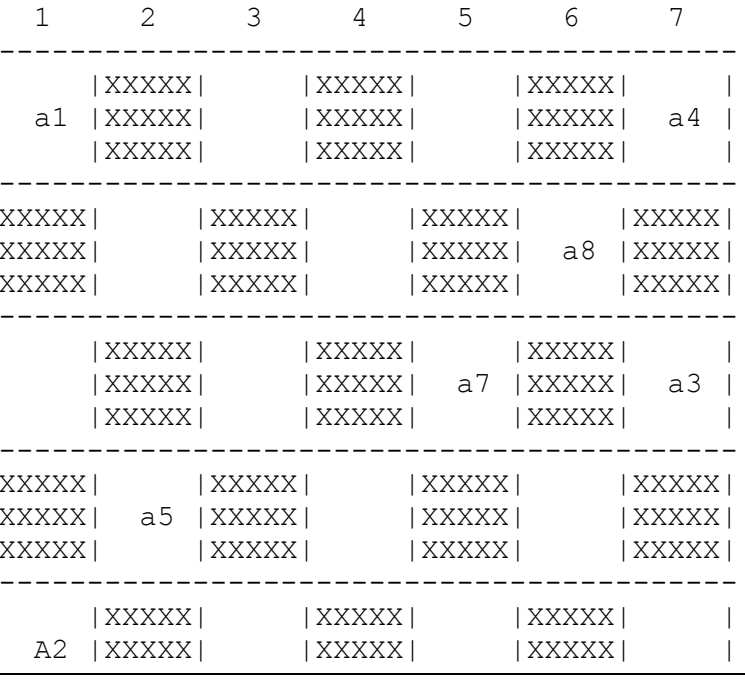
A. solutions that ask the user to input the row and column of the piece to be removed.</td><td>9</td></tr>
</table>

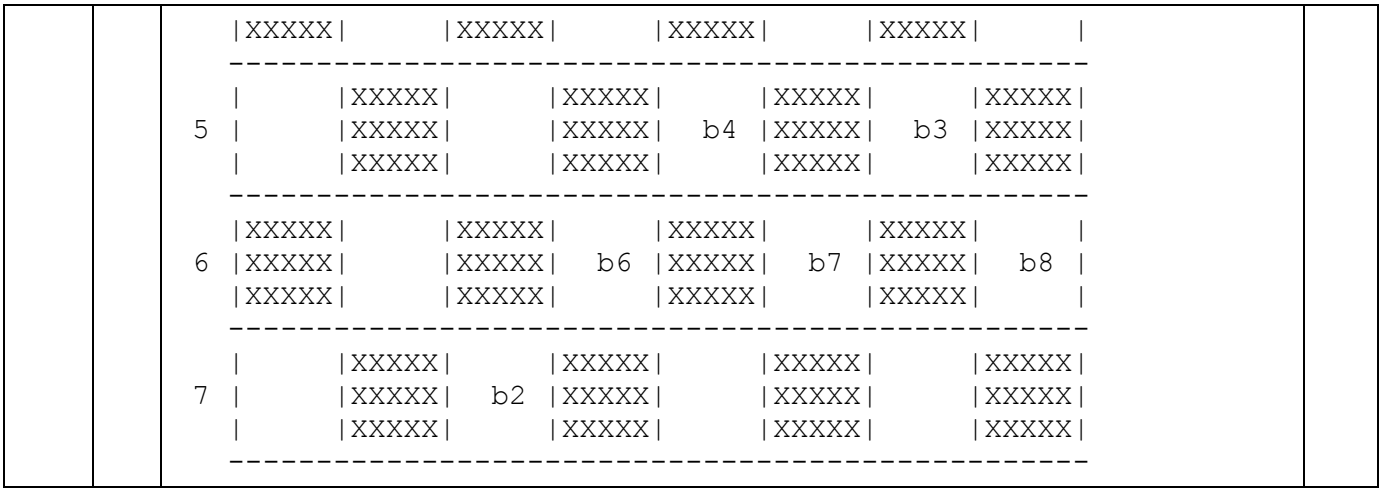
9	**3**	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 17.2, including prompts on screen capture matching those in code.
		Code for 17.2 must be sensible.
		
		Screen capture showing:
		`Do you want to load a saved game? (Y/N): y`
		`Enter the filename: game3.txt`
		
		`Player A:`
		`[[8, 0, 0], [0, 1, 0], [6, 1, 0], [2, 7, 0], [0, 7, 0], [3, 2, 0], [3, 0, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]`
		`Player B:`
		`[[8, 0, 0], [4, 1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [1, 4, 0], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]`
		
		`  0  1  2  3  4  5  6  7`
		`0  a1  a4`
		`1  b5  a8`
		`2  a7  a3`
		`3  a6  a5`
		`4  b1`
		`5  b4  b3`
		`6  a2  b6  b7  b8`
		`7  b2`
		
		`Next Player:  a`
		`a1  can move to  1  ,  0`
			1

```
a1  can move to  1  ,  2
a2  can move to  7  ,  0
a3  can move to  3  ,  6
a5  can move to  4  ,  3
a7  can move to  3  ,  4
a7  can move to  3  ,  6
a8  can jump to  3  ,  4
There are  8  possible moves
Which piece do you want to move? a2
Which row do you want to move to? 7
Which column do you want to move to? 0
Which piece do you want to take? b1

Player A:
[[9, 1, 0], [0, 1, 0], [4, 1, 1], [2, 7, 0], [0, 7, 0], [3, 2,
0], [3, 0, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0],
[-1, -1, 0], [-1, -1, 0]]
Player B:
[[8, 0, 0], [-1, -1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [1,
4, 0], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1,
0], [-1, -1, 0], [-1, -1, 0]]

     0     1     2     3     4     5     6     7
   -----------------------------------------------
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
 0 |XXXXX|  a1 |XXXXX|     |XXXXX|     |XXXXX|  a4 |
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
   -----------------------------------------------
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
 1 |     |XXXXX|     |XXXXX|  b5 |XXXXX|  a8 |XXXXX|
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
   -----------------------------------------------
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
 2 |XXXXX|     |XXXXX|     |XXXXX|  a7 |XXXXX|  a3 |
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
   -----------------------------------------------
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
 3 |  a6 |XXXXX|  a5 |XXXXX|     |XXXXX|     |XXXXX|
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
   -----------------------------------------------
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
 4 |XXXXX|  A2 |XXXXX|     |XXXXX|     |XXXXX|     |
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
   -----------------------------------------------
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
 5 |     |XXXXX|     |XXXXX|  b4 |XXXXX|  b3 |XXXXX|
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
   -----------------------------------------------
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
 6 |XXXXX|     |XXXXX|  b6 |XXXXX|  b7 |XXXXX|  b8 |
   |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|     |
   -----------------------------------------------
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
 7 |     |XXXXX|  b2 |XXXXX|     |XXXXX|     |XXXXX|
   |     |XXXXX|     |XXXXX|     |XXXXX|     |XXXXX|
   -----------------------------------------------
```

9	**4**	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** Must match code from 17.2, including prompts on screen capture matching those in code. Code for 17.2 must be sensible. Screen capture showing: Next Player: b b2 can move to 6 , 1 b3 can move to 4 , 5 b3 can move to 4 , 7 b4 can move to 4 , 3 b4 can move to 4 , 5 b5 can move to 0 , 3 b5 can move to 0 , 5 b6 can move to 5 , 2 b6 can jump to 4 , 5 b7 can jump to 4 , 3 b7 can jump to 4 , 7 b8 can jump to 4 , 5 There are 12 possible moves Which piece do you want to move? b5 Which row do you want to move to? 0 Which column do you want to move to? 3 Which piece do you want to take? a6 Player A: [[9, 1, 0], [0, 1, 0], [4, 1, 1], [2, 7, 0], [0, 7, 0], [3, 2, 0], [-1, -1, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]] Player B: [[9, 1, 0], [-1, -1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [3, 0, 1], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]  	**1**


5
6
7
b4
b3
b6
b7
b8
b2

10	**1**	**Mark is for AO1 (understand)** `Header` / `FileHeader`; **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing	**1**

10	**2**	**Mark is for AO1 (understand)** `Grid / Fields;` **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing	**1**

11		**1 mark for AO1 (knowledge) and 1 mark for AO2 (analyse)** Identifier (AO2): `OK` / `InputError` / `Finished` / `Solved` / `Correct` / `Incomplete`; Data type (AO1): `Boolean` / `bool`; **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing	**2**
12	**1**	**Mark is for AO2 (analyse)** (Stores the string equivalent of User's) score;	**1**
12	**2**	**3 marks for AO2 (analyse)** 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); **Max 1** 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; **Max 2**	**3**

13		**2 marks for AO3 (design)** Row/Column heading numbers need to be extended; Row/Column headers formatted to occupy two positions; Sub-grid lines need to extended horizontally; **I.** vertically Grid lines need to extended horizontally; **I.** 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; **R.** reference to alterations outside the `DisplayGrid` subroutine. **Max 2**	**2**

14	**1**	**Mark is for AO1 (understand)** `AssemblerInstruction`; **A.** `Memory` / `SymbolTable`; **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing **Max 1**	**1**
14	**2**	**Mark is for AO1 (understand)** `SourceCode` / `Registers` / `OpCodeValues`; **A.** `Memory` / `SymbolTable`; **(if neither given in 07.1)** **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing **Max 1**	**1**

Qu		Marks	
15	1	**Mark is for AO2 (analyse)** `BuyerQ`; **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing	**1**
15	2	**Mark is for AO2 (analyse)** `Stats` / `Tills` / `Data`; **A**. `buyerInfo` (Java only) **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing **Max 1**	**1**

Qu		Marks	
16	1	**Mark is for AO2 (analyse)** Divides `Stats[TOTAL_Q]` by `Stats[TOTAL_Q_OCCURENCE]`;	1

Qu		Marks	
16	2	**2 marks for AO2 (analyse)** `Stats[TOTAL_Q]` has the length of the queue added onto it (in each time unit that there is a non-empty queue); `Stats[TOTAL_Q_OCCURRENCE]` is incremented in each time unit that there is a (non-empty) queue; **DPT** within 14.2 reference to an index such as `TOTAL_Q` rather than the data to which it points, such as `Stats[TOTAL_Q]`	**2**
17		**2 marks for AO3 (design)** Need a 2D data structure/list of lists to store the queues // add a field to `Q_Node` to store which till the buyer is queuing for // add a queue/array/list to each `Till`/element of the `Tills` data structure; **R.** Use one array per `Till` Need code to allocate buyers to different queues // the code that moves someone out of the queue to be served would need changing // the code that moves everyone in the queue up would need changing // the code that displays the contents of the queue would need changing;	**2**