01	**1**	**Mark is for AO1 (understanding)** `Orientation`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
01	**2**	**Mark is for AO1 (understanding)** `CheckWin` // `PrintBoard` // `SetUpBoard` // `SetUpShips`; **R.** if any additional code (including routine interface) **R.** if spelt incorrectly **I.** case	1
01	**3**	**Mark is for AO1 (understanding)** `ValidateBoatPosition` // `CheckWin`; **R.** if any additional code (including routine interface) **R.** if spelt incorrectly **I.** case	1

02	**1**	**Mark is for AO1 (understanding)** Improves readability of code; Easier to update the programming code if the value changes (**A**. by implication); Reduce the likelihood of inconsistency causing errors; Unlike a variable the value can't be changed / is not mutable; **Max 1**	1
02	**2**	**All marks AO2 (analyse)** **1 mark:** `FOR` loop is used to run across the length of the new ship // Number of iterations/length of ship is known; **1 mark:** program checks if the board already has a ship positioned in this cell // this cell is not empty; **1 mark:** (Use of loop structure) avoids the need for many `IF` statements; **Max 2**	2
02	**3**	**All marks AO2 (analyse)** Code is checking that the end of a ship does not go outside of the board / outside of bounds of array / greater than 10; for a ship placed vertically; **A.** Boat does not go off the <u>bottom</u> of the board ;;	2
02	**4**	**1 mark for AO1 (knowledge) and 2 marks for AO2 (analyse)** **AO1 1 mark:** Exception handling is responding to the occurrence of fatal errors / errors that would cause a crash / runtime errors; **AO2 1 mark:** Could be used to trap any errors when converting user input into an integer // to trap an error if a non-integer entry; **AO2 1 mark:** Code could then ask user to re-enter the value // default value used // automatically count as a miss // display an error message;	3

03	**1**	**Mark is for AO2 (analysis)** `MakePlayerMove`; **R.** if spelt incorrectly **I.** case	1
03	**2**	**Mark is for AO2 (analysis)** `CheckWin`; **R.** if spelt incorrectly **I.** case	1
03	**3**	**Mark is for AO2 (apply)** `GetRowColumn`; **R.** if spelt incorrectly **I.** case	1
03	**4**	**All marks AO1 (understanding)** Because the scope of the two variables is different ;; // Because they are (both) local variables; in different subroutines;	2
03	**5**	**All marks AO1 (understanding)** In text files all data is stored as strings / ASCII values / Unicode values / characters // text files use only byte values that display sensibly on a VDU // stores only values that can be open and read in a text editor; **A.** text file is human-readable Binary files store data using different data types; **A.** By example **A.** Binary files can only be correctly interpreted by application that created them	2

04	1	**1 mark for AO3 (design) and 4 marks for AO3 (programming)** Note that AO3 (design) marks 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. **AO3 (design) - 1 mark:** **1.** Identifying that an appropriate technique is required to repeatedly input the data; **AO3 (programming) - 4 marks:** **2.** Check against lower/upper bound for `Row`; **3.** 2. combined correctly with check against other bound; **4.** The message "`Invalid value entered`" is displayed only if invalid value has been entered for at least one of the correct bounds; **I.** Case of output message **A.** Minor typos in output message **I.** Spacing in output message **5.** Function only returns values if row is valid and always returns a value if row is valid; **NOTE:** 2. and 3. could be combined into checking against a range of values and be awarded both marks **NOTE:** Ignore any code validating against column	5

POSSIBLE ANSWER

```
Console.WriteLine();
Console.Write("Please enter column: ");
Column = Convert.ToInt32(Console.ReadLine());
do
{
        Console.Write("Please enter row: ");
        Row = Convert.ToInt32(Console.ReadLine());
        if (Row < 0 || Row > 9)
        {
                Console.WriteLine("Invalid value entered");
        }
} while (Row < 0 || Row > 9);
Console.WriteLine();
```

04	**2**	**Mark is for AO3 (evaluate)**
		
		****SCREEN CAPTURE****
		Must match code from 09.1, including prompts on screen capture matching those in code. Code for 09.1 must be sensible.
		
		`Please enter column: 6`
		`Please enter row: 10`
		
		`Invalid value entered`
		
		`Please enter column: 6`
		`Please enter row: 9`
		
		`Hit at (6,9)`
		
		Mark as follows
		Screen capture(s) showing the requested test being performed;
		
		NOTE: if they have coded a check to validate column the screenshot might not show the second request for entry of column as 6 would be seen as valid
			1

05	**1**	**2 marks for AO3 (design) and 6 marks for AO3 (programming)**	8

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. The `Ships` data structure is modified correctly for each hit. The sunk message is displayed at the appropriate time. To award eight marks, the code must perform exactly as required in the question. It is evident from the program code that the code has been designed appropriately to ensure that the task is achieved.	7-8
3	There is evidence that a line of reasoning has been followed to produce a logically structured subroutine that works correctly in most cases but with some omissions (e.g. the `Ships` data structure is modified incorrectly or the message that is displayed may not match the question). It is evident from the program code that it has been designed appropriately to ensure that the task is mainly achieved.	5-6
2	There is evidence that a line of reasoning has been partially followed as the `Ships` data structure is modified (though possibly incorrectly). The correct message might not be displayed. There is little or no evidence that a line of reasoning has been followed to award a mark for the design of the solution.	3-4
1	An attempt to modify the `Ships` data structure. This modification may not be in exactly the right place and the value to change the structure by may be incorrect, but it should be possible to see that it was intended to be linked to a particular ship. To award two marks instead of one, some of the code must be syntactically correct. There is insufficient evidence to suggest that a line of reasoning has been followed or that the solution has been designed.	1-2

Guidance

Evidence of AO3 (design) - 2 points:

1. designing a suitable interface for the `CheckSunk` subroutine with suitable parameters and return values;

   Suitable parameters could be `board,ships,row,column` OR `ships` and the character from `board[row,column]`;

2. identifying suitable structures so that every ship can be checked (e.g. loop or 5 `IF` statements);

Note that 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) - 6 points:

3. `CheckSunk` subroutine created with begin and end of subroutine;

4. Use the letter stored in the square that has been hit;
   ie use of `board[row][column]` and set to a variable or used with a comparison to a ship name first character

5. Compare the letter from the square (4.) with the first letter of the type of ship ;
   ie either via direct characters such as '`A`','`P`' or using something similar to `ships[][][0]`
   R. if would only work for 2 or fewer ships

6. Take one away from the size of the ship identified as being hit;
   R. if would only work for 2 or fewer ships

7. Selection statements to compare size of the hit ship against 0;
   R. if would only work for 2 or fewer ships

8. Correct message (type of ship followed by `'is sunk!'`) is displayed under the correct circumstances for **all** ships;
   NOTE: For the 'sunk' message look to make sure it is only being outputted at the time of sinking

   I. Case of output message
   A. Minor typos in output message
   I. Spacing in output message

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

NOTE: Code could be written as a function or procedure - either would be acceptable.

05	**1**	private static void CheckSunk(int Row, int Column, char[,] Board, ref ShipType[] Ships)
		{
		for (int i = 0; i < 5; i++)
		{
		if (Ships[i].Name[0] == Board[Row, Column])
		{
		Ships[i].Size = Ships[i].Size - 1;
		if (Ships[i].Size == 0)
		{
		Console.WriteLine(Ships[i].Name + " is sunk!");
		}
		}
		}
		}
		Alternative (use of if / case / select for each ship type):
		private static void CheckSunk(int Row, int Column, char[,] Board, ref ShipType[] Ships)
		{
		int ShipIndex;
		if (Board[Row,Column] == 'A')
		{
		ShipIndex=0;
		}
		else if (Board[Row, Column] == 'B')
		{
		ShipIndex=1;
		}
		else if (Board[Row, Column] == 'S')
		{
		ShipIndex=2;
		}
		else if (Board[Row, Column] == 'D')
		{
		ShipIndex=3;
		}
		else if (Board[Row, Column] == 'P')
		{
		ShipIndex=4;
		}
		Ships[ShipIndex].Size -= 1;
		if (Ships[ShipIndex].Size == 0)
		{
		Console.WriteLine(Ships[ShipIndex].Name + " is sunk!");
		}
		}
			8

05	**2**	**Mark is for AO3 (programming)** **1 mark:** Call to `CheckSunk` is in the correct position in the `MakePlayerMove` subroutine (must be before setting the board cell to `'h'`); **A.** if they have declared a variable to temporarily store the current `board[row][column]` ship character and passed this before or after setting board cell to `'h'`	1

```
...
else
{
 Console.WriteLine("Hit at (" + Column + "," + Row + ").");
 CheckSunk(Row, Column, Board, ref Ships);
 Board[Row, Column] = 'h';
...
```

05	**3**	**Mark is for AO3 (evaluate)** *****SCREEN CAPTURE***** *Must match code from 10.1 and 10.2, including prompts on screen capture matching those in code. Code for 10.1 and 10.2 must be sensible.* `Please enter column: 1` `Please enter row: 5` `Hit at (1,5)` `Patrol Boat is sunk!` `The board looks like this:` `  0   1   2   3   4   5   6   7   8   9` `0								|` `1								|` `2								|` `3	m							|` `4	h							|` `5	h							|` `6								|` `7								|` `8								|` `9								|`  **Mark as follows** Screen capture(s) showing the final shot being performed (1, 5), final board layout and the message `'Patrol Boat is sunk!'`; **I**. Message concerning firing a torpedo (candidate may have attempted question 11 before 10)	1

06	**1**	**2 marks for AO3 (design) and 10 marks for AO3 (programming)** Note that AO3 (design) marks 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.	12

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that is efficient. Code is written to ensure only one torpedo can be fired during a game and offers the user a chance to fire a torpedo if appropriate. When a torpedo is fired it moves correctly. Appropriate messages are displayed. A formal interface is used to pass at least some of the required data into and out of the `MakePlayerTorpedoMove` subroutine. All of the appropriate design decisions have been taken.	10-12
3	There is evidence that a line of reasoning has been followed. The `PlayGame` subroutine has been altered to allow the user to fire a torpedo. The `MakePlayerTorpedoMove` subroutine uses a logically structured subroutine that works correctly in most cases. A formal subroutine interface may or may not have been used. The solution demonstrates good design work as most of the correct design decisions have been taken.	7-9
2	The `PlayGame` subroutine has been adapted but it might not ensure that the player can only fire one torpedo. A `MakePlayerTorpedoMove` subroutine structure has been created and there may be some appropriate, syntactically correct programming language statements written. There is evidence that a line of reasoning has been partially followed for `PlayGame` and/or `MakePlayerTorpedoMove` as although there may not be all of the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4-6
1	An attempt has been made to alter the `PlayGame` subroutine and/or a subroutine has been created and some appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct and the subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-3

AO3 (design) - 2 marks:

1. Identification of the need for a variable to hold whether a torpedo has been fired or not;

2. Use of an iteration structure to control the moving of the torpedo;

AO3 (programming) - 10 marks:

PlayGame

3. Logic to determine whether a torpedo can be fired is correct and is in an appropriate place in the `PlayGame` subroutine;

4. Correct prompt '`Fire a torpedo? (Y/N)`' is displayed;

   I. Case of output message
   A. Minor typos in output message
   I. Spacing in output message

5. Selection statement coded to handle a torpedo shot or a normal shot calling the relevant subroutine and code is in appropriate place in the `PlayGame` subroutine;

6. Variable is correctly updated to reflect the fact that a torpedo has been fired;

MakePlayerTorpedoMove

7. A suitable interface for the `MakePlayerTorpedoMove` subroutine with suitable parameters (`board` and possibly `ships`) and return value

8. Code correctly exits iteration when torpedo moves off board and torpedo hits a ship

9. Code will move torpedo up board

10. Code correctly determines when torpedo has hit a ship updating board to '`h`' (when necessary)

11. If current board position is empty cell '`-`' is updated to contain an '`m`' and code behaves correctly for cell already containing an '`m`'.

12. Messages produced appropriately and displayed under correct conditions when the torpedo misses (moves off the board) and hits a ship.

NOTE: `MakePlayerTorpedoMove` could be written as a procedure or function - either would be acceptable

NOTE: A call to `CheckSunk` is not necessary for full marks

06	1	private static void PlayGame(ref char[,] Board, ref ShipType[] Ships)
		{
		bool GameWon = false;
		bool TorpedoUsed = false;
		string TorpedoChosen;
		while (GameWon == false)
		{
		PrintBoard(Board);
		if (TorpedoUsed == false)
		{
		Console.WriteLine("Fire a torpedo? (Y/N)");
		TorpedoChosen = Console.ReadLine();
		}
		if (TorpedoChosen == "Y")
		{
		MakePlayerTorpedoMove(ref Board, ref Ships);
		TorpedoChosen = "N";
		TorpedoUsed = true;
		}
		else
		{
		MakePlayerMove(ref Board, ref Ships);
		}
		GameWon = CheckWin(Board);
		if (GameWon == true)
		{
		Console.WriteLine("All ships sunk!");
		Console.WriteLine();
		}
		}
		}
		
		private static void MakePlayerTorpedoMove(ref char[,] Board, ref ShipType[] Ships)
		{
		int Row = 0;
		int Column = 0;
		GetRowColumn(ref Row, ref Column);
		while (Row > 0 && (Board[Row, Column] == 'm' || Board[Row, Column] == '-'))
		{
		Board[Row, Column] = 'm';
		Row = Row - 1;
		}
		if (Board[Row, Column] <> '-' && Board[Row, Column] <> 'm')
		{
		Console.WriteLine("Torpedo hits at (" + Column + "," + Row + ").");
		Board[Row, Column] = 'h';
		}
		else
		{
		Console.WriteLine("Torpedo failed to hit a ship.");
		Board[Row, Column] = 'm';
		}
		}
			12

06	2	**All marks for AO3 (evaluate)** *****SCREEN CAPTURE***** *Must match code from 11.1, including prompts on screen capture matching those in code. Code for 11.1 must be sensible.*	2

```
MAIN MENU

1. Start new game
2. Load training game
9. Quit

Please enter your choice: 2

The board looks like this:

  0   1   2   3   4   5   6   7   8   9
0   |   |   |   |   |   |   |   |   |
1   |   |   |   |   |   |   |   |   |
2   |   |   |   |   |   |   |   |   |
3   |   |   |   |   |   |   |   |   |
4   |   |   |   |   |   |   |   |   |
5   |   |   |   |   |   |   |   |   |
6   |   |   |   |   |   |   |   |   |
7   |   |   |   |   |   |   |   |   |
8   |   |   |   |   |   |   |   |   |
9   |   |   |   |   |   |   |   |   |

Fire a torpedo(Y/N)? Y

Please enter column: 1
Please enter row: 7

Torpedo hits at (1,5)

The board looks like this:

  0   1   2   3   4   5   6   7   8   9
0   |   |   |   |   |   |   |   |   |
1   |   |   |   |   |   |   |   |   |
2   |   |   |   |   |   |   |   |   |
3   |   |   |   |   |   |   |   |   |
4   |   |   |   |   |   |   |   |   |
5   | h |   |   |   |   |   |   |   |
6   | m |   |   |   |   |   |   |   |
7   | m |   |   |   |   |   |   |   |
8   |   |   |   |   |   |   |   |   |
9   |   |   |   |   |   |   |   |   |
```

```
Please enter column: 2
Please enter row: 1

Sorry, (2,1) is a miss.

The board looks like this:

  0   1   2   3   4   5   6   7   8   9
0   |   |   |   |   |   |   |   |   |
1   |   | m |   |   |   |   |   |   |
2   |   |   |   |   |   |   |   |   |
3   |   |   |   |   |   |   |   |   |
4   |   |   |   |   |   |   |   |   |
5   | h |   |   |   |   |   |   |   |
6   | m |   |   |   |   |   |   |   |
7   | m |   |   |   |   |   |   |   |
8   |   |   |   |   |   |   |   |   |
9   |   |   |   |   |   |   |   |   |
```

Mark as follows
1 mark: Screenshots show player is asked if they want to place a torpedo and then the '`Fire a torpedo?`' prompt is not shown for the second shot;

1 mark: Screenshots show the torpedo moves correctly from (1,7) to (1,5) and causes a hit;

NOTE: Do not penalise extra screenshots showing the torpedo moving up square by square