DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY
COURSEWORK ASSESSMENT DESCRIPTION 2020/21

UNIVERSITY OF HULL

MODULE DETAILS:

Module Number:	400068	Trimester:	1
Module Title:	Programming and Algorithmic Thinking		
Lecturer:	Simon Grey		

COURSEWORK DETAILS:

Assessment Number:	1	of	3		
Title of Assessment:	Wordsearch Application				
Format:	Software Application				
Method of Working:	Individual				
Workload Guidance:	Typically, you should expect to spend between	40	and	60	hours on this assessment
Length of Submission:	This assessment should be **no** more than: *(over length submissions **may have only** **parts of the work marked** as per University policy)*	**N/A words** *(excluding diagrams, appendices, references, code)*			

PUBLICATION:

Date of issue:	06/11/2020

SUBMISSION:

ONE copy of this assessment should be handed in via:	Canvas		If Other (state method)	GitHub
Time and date for submission:	**Time**	2pm	**Date**	Thursday 17th December
If **multiple hand–ins** please provide details*:*	The date listed is the final hand in. Students are encourage to submit to the draft assessment by Wednesday 3rd December in order to get formative feedback prior to the hand in. Summative feedback will be provided via rubric only.			
Will submission be scanned via TurnitinUK?	No	For Turnitin, these should be one of the allowed types e.g. Word, RT, PDF, PPT, XLS etc. Specify any particular requirements in the submission details on TurnItIn. Unless specified: students MUST NOT submit ZIP or other archive formats unless specified. Students can **ONLY** submit **ONE** file and must ensure they upload the correct file. Normally only the LAST submission will be considered – the last submission is late it should incur a late penalty.		

The assessment must be submitted **no later** than the time and date shown above, unless an extension has been authorized.

MARKING:

Marking will be by:	Student Name

ASSESSMENT:

The assessment is marked out of:	100	and is worth	40	% of the module marks
N.B If multiple hand-ins please indicate the marks and % apportioned to each stage above (i.e. Stage 1 – 50, Stage 2 – 50). It is these marks that will be presented to the exam board.				

ASSESSMENT STRATEGY AND LEARNING OUTCOMES:

The overall assessment strategy is designed to evaluate the student's achievement of the module learning outcomes, and is subdivided as follows:

LO	Learning Outcome	Method of Assessment *{e.g. report, demo}*
LO1	Identify problems which are amenable to computer based solutions and suggest how such a solution can be structured and deployed.	Software application
LO2	Systematically analysesimple data processing problems and express solutions in an object oriented programming language.	Software application
LO3	Apply knowledge of the syntax and semantics of a specific programming language.	Software application
LO4	Make use of appropriate tools to create and deploy simple programs in a specific programming language.	Software application

Assessment Criteria	Contributes to Learning Outcome	Mark
Wordsearch specific functionality as described below	1 2 3 4	30%
General code quality includes use of appropriate datastructures, methods, exception handling, readability and maintainability of code and use of source control Note: A more detailed rubric is available on Canvas	1 2 3 4	70%

FEEDBACK

Feedback will be given via:	Canvas Rubric	Feedback will be given via:	Select secondary method

Exemption (staff to explain why)	
Feedback should be provided no later than 4 ‘teaching weeks’ after the submission date.	

This assessment is set in the context of the learning outcomes for the module and does not by itself constitute a definitive specification of the assessment. If you are in any doubt as to the relationship between what you have been asked to do and the module content you should take this matter up with the member of staff who set the assessment as soon as possible.

You are advised to read the **NOTES** regarding late penalties, over-length assignments, academic misconduct and quality assurance in your student handbook, which is available on Canvas.

In particular, please be aware that:

- Up to and including 24 hours after the deadline, a penalty of 10%
- More than 24 hours and up to and including 7 days after the deadline; either a penalty of 10% or the mark awarded is reduced to the pass mark, **whichever results in the lower mark**
- More than 7 days after the deadline, a mark of zero is awarded.
- The overlength penalty applies to your written report (which includes bullet points, and lists of text. It does not include contents page, graphs, data tables and appendices). Work beyond the specififed length may not be marked.
- You are responsible for reading the University Code of Practice on Academic Misconduct. This govern all forms of illegitimate academic conduct which may be described as cheating, including plagiarism.

In case of any subsequent dispute, query, or appeal regarding your coursework, you are reminded that it is your responsibility to produce the assignment in question.

400068 Introduction to Programming and Algorithmic Thinking

Assessed Coursework : Wordsearch

Introduction

This is the coursework specification for the 'Introduction to Programming and Algorithmic Thinking' Assessed Coursework. More information has been given in module content which will be invaluable in completing this coursework satisfactorily.

Marks

This coursework is worth 40 % of the module. The Wordsearch Application is marked out of 100. The marking rubric is available as part of the assignment on canvas.

Submission Date and Feedback

There are two submissions for this piece of work. This is to enable us to give you feedback on your work before the final deadline to give you the opportunity to use our feedback to improve your work.

The first draft submission is on **Wednesday 2nd December at 2pm**. Work submitted by this date will be marked using a detailed rubric and will receive written feedback indicating what aspects may be improved. Students may also request feedback on specific areas of their work as part of their submission. Feedback will be delivered by **Wednesday 9th December** giving student time to act upon their feedback prior to the final deadline.

The final deadline is **Thursday 17th December at 2pm**. After this final deadline students will receive their final mark generated using the same marking rubric. There will be no additional feedback beyond that given through the rubric at this stage.

GitHub Classroom

The main submission mechanism is via GitHub classroom. An assignment will be created in GitHub classroom. When you accept the assignment a repository will be created for you with a base .net Core Console Application project and the associated test files. When you are ready to submit your work you should tag the file "draft submission" for the draft submission, and "final submission" for the final submission. Details on how to do this will be included in the canvas assignment.

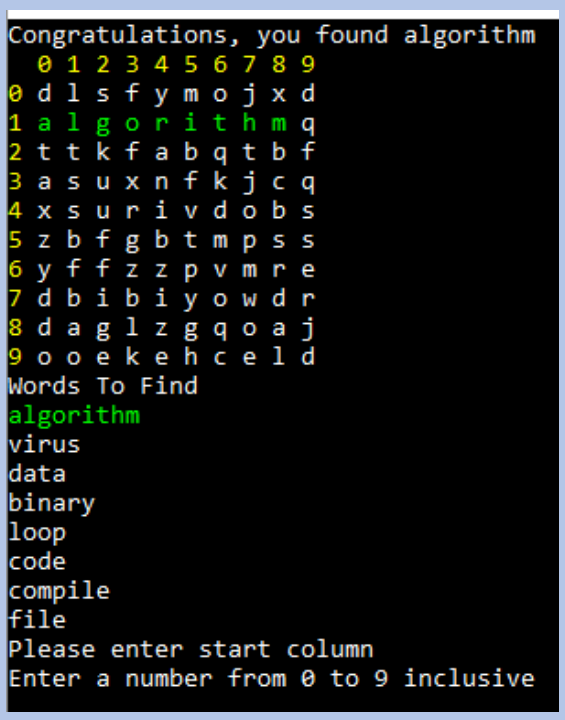
You must also back up your submission to ensure it has been correctly uploaded by submitting your solution as a .zip file in Canvas. You should zip and submit the whole solution folder to Canvas for marking including the .sln file, .proj file and .cs file. There are marks available for your appropriate use of GitHub so ensure you are regularly checking in your changes and using it appropriately.

Accept the assignment by clicking on this link

https://classroom.github.com/a/AwHDoHOf

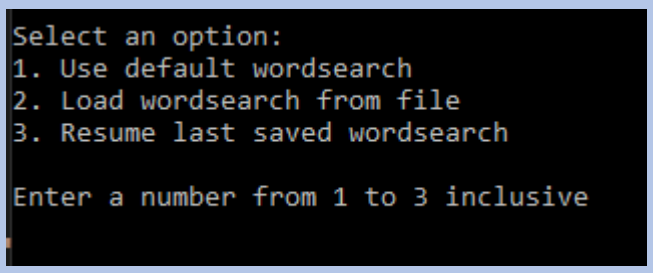
Wordsearch Application

A word search puzzle consists of letters arranged in a grid, some of which form words. The player solves the puzzle by finding all these (hidden) words which can be displayed in any direction. An example of a wordsearch is shown below.



For this ACW the student must write a C# console application that implements a word search. The steps that follow form a specification of how the program should function and its features. Where the specification is vague the student can make assumptions, however, the program should not have any features additional to what is described in this specification. All assumption should be clarified with a member of the module staff.

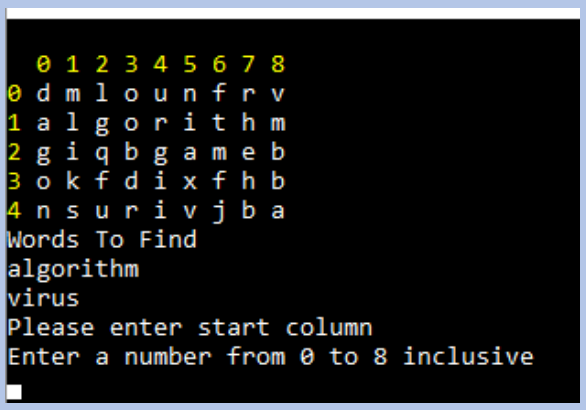
1. Building a Wordsearch



B should be presented with a menu with options to either use a default puzzle file, load the current saved wordsearch or select from a list of files.

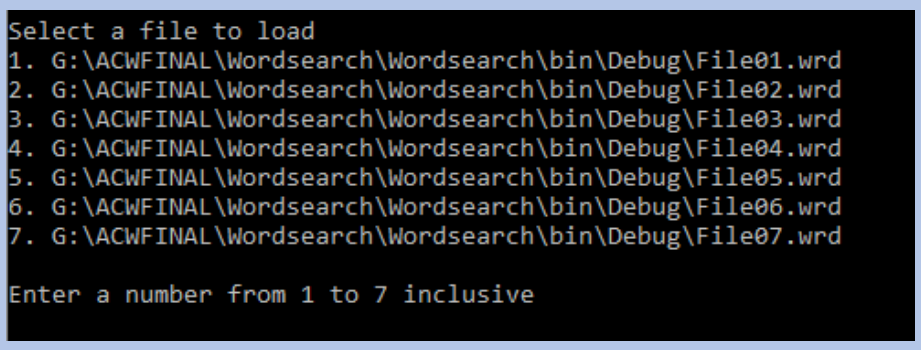
2. Using the default puzzle file

If the user selects the first option from the initial menu they should be presented with a default wordsearch that is the same as is found in “File01.wrd” but is ***hard coded into the application***. This will ensure students are still able to make progress if they struggle to load wordsearch puzzles from files.



3. Selecting from a list of files

If a user selects the second option the program should display ***all files in the current directory*** with the file extension ".wrd".



Note – this means that your program should look in the current directory to see what file are in the current directory, and ask the user to pick from them.

The following code can be used to return an array of strings with the appropriate file paths:

```
string[] files = Directory.GetFiles(Directory.GetCurrentDirectory(), "*.wrd");
```

The solution that you have been provided with includes seven wordsearch files. Be aware that some are provided as test files that your program should reject for various reasons – being in the wrong format for example. The program should be able to load any file that is in the correct format. If the program attempts to load a file in the incorrect format it should display an error message and give the user the option to load another file or use the default file. Details of the file format can be found later in this document in the section on "Puzzle File Specification".

Some example test files have been provided for you. You may wish to add additional test cases to your solution. Details of the provided test files are in the table below.

Filename	**Comments**
File01.wrd	contains only horizontal words
File02.wrd	contains only horizontal and vertical words
File03.wrd	contains horizontal, vertical and diagonal words
File04.wrd	Incorrect number of words
File05.wrd	Incorrect File Format
File06.wrd	Words outside of grid
File07.wrd	Incorrect overlapping words

As a file is parsed the word search data should be generated and stored in variables. The program should make use of appropriate data structures such as arrays and structs to store the word search

Puzzle File Specification

The word search puzzles are defined in a file that is loaded and parsed by the program. The file uses a comma-separate variable (CSV) format.
The first line of the file specifies:

- *<Number of Columns>, <Number of Rows>, <Total Number of hidden words>*
- Each line that follows specifies the word and its properties:
- *<Word>, <Column Co-ordinate of first letter>, <Row Co-ordinate first letter>, <Direction>*
- Here is an example of a puzzle file;

```
6,5,3
apple,0,0,right
pear,1,0,down
orange,0,3,right
```

- 6x5 puzzle grid with a total of 3 hidden words
- Word ‘apple’ starts at cell (0,0) and reads in the right-ward direction
- Word ‘pear’ starts at cell (1,0) and reads in the down-ward direction
- Word ‘orange’ starts at cell (0,3) and reads in the right-ward direction
- Here is a graphical representation of what the file specifies;
- Note that the origin (0,0) is at the top-left cell

4. Loading a Partially Complete Wordsearch

- If the user chooses to Resume Wordsearch the program should attempt to load the file "wordsearch.sav"
- The format of the file is undefined, but should encapsulate all the state of the wordsearch such that a user could continue their wordsearch from the point when the program was saved

5. Display the word search board and list the hidden words

- The puzzle board should have the correct dimensions as specified in the puzzle file
- The perimeter axis should be labelled and highlighted Yellow
- The hidden words are displayed on the board. The remainder of the board is filled with random letters
- The program must also display, beneath the board, a list of the hidden words that have yet to be found

6. Prompt user for input

- The user should be asked to either identify a word or save and quit.
- The program must prompt the user to identify a word by inputting 2 co-ordinates, one for the first letter and one for the last letter
- In the current example the word 'apple' can be identified by the co-ordinates (0,3) for the first letter and (4,3) for the last letter.
- Note that the order is matters i.e. inputting (4,3) for the first letter and (0,3) for the last letter is incorrect as it spells 'elppa'
- The user input should be validated and if incorrect the program should handle it (i.e. not crash) and prompt the user of their mistake.

7. Process input

- The program should process the user's input and detect if a word is found.
- The program data should be updated to reflect that a word has been found
- The program should give the user the option to save their progress and quit

8. Saving Progress

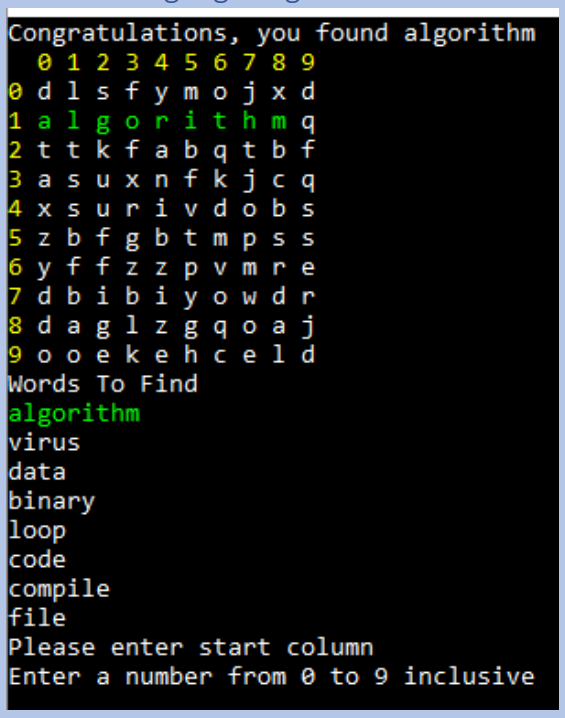
- If the user chooses to save their progress you should save the to a file called "wordsearch.sav". Only one ongoing wordsearch needs to be saved as a time
- The format of the file is undefined, but should encapsulate all the state of the wordsearch such that a user could continue their wordsearch from the point when the program was saved
- After saving the program should close

9. Repeat steps 5-7 until all words have been found

- The program should detect when there are no more words to be found
- Display a message congratulating the user
- The user should be given the option to load a new wordsearch or quit the program

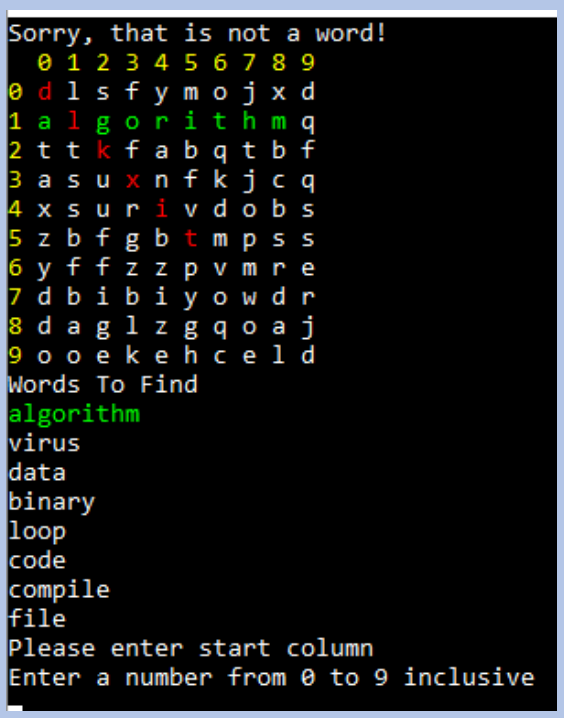
```
  0 1 2 3 4 5
0 a p p l e d
1 l e s y m l
2 t a y j m d
3 o r a n g e
4 v r m f s t
Words To Find
apple
pear
orange
Please enter start column
Enter a number from 0 to 5 inclusive
0
Please enter start row
Enter a number from 0 to 4 inclusive
3
Please enter end column
Enter a number from 0 to 5 inclusive
5
Please enter end row
Enter a number from 0 to 4 inclusive
3
Congratulations, you found all the words!
Press any key to exit.
```

Program Enhancement – Colour Highlighting



When displaying the puzzle board all the words that have been found should be coloured Green

If the user's makes an incorrect selection it should be coloured Red. Only the previous incorrect selection is highlighted i.e. the red highlighting does not accumulate as the user makes more incorrect selections



ACW Puzzle Files

Here are the details about the 3 files that the program will need to parse for the ACW. These will be made available to you.

File01.txt

```
9,5,2
algorithm,0,1,right
virus,5,4,left
```

```
  0 1 2 3 4 5 6 7 8
0 j y a m h r e w p
1 a l g o r i t h m
2 o z a l m m r u z
3 j s m n e r q p v
4 o s u r i v x v g
Words To Find
algorithm
virus
Please enter start column
Enter a number from 0 to 8 inclusive
```

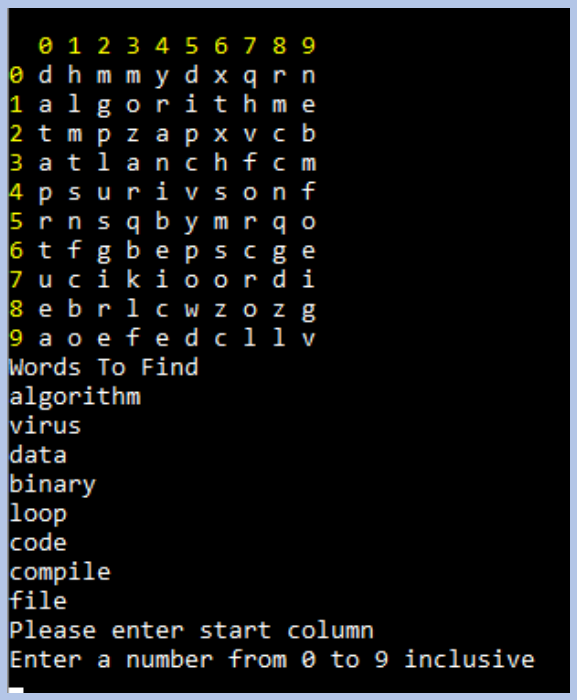
File02.txt

```
9,6,4
algorithm,0,1,right
virus,5,4,left
data,0,0,down
binary,4,5,up
```

```
  0 1 2 3 4 5 6 7 8
0 d n s z y u p s i
1 a l g o r i t h m
2 t v b q a g m z u
3 a v h a n b e u y
4 o s u r i v q c b
5 n m l l b l m a x
Words To Find
algorithm
virus
data
binary
Please enter start column
Enter a number from 0 to 8 inclusive
```

File03.txt

```
10,10,8
algorithm,0,1,right
virus,5,4,left
data,0,0,down
binary,4,5,up
loop,8,9,leftup
code,6,9,rightup
compile,8,3,leftdown
file,1,6,rightdown
```



Additional Test Files

You should not assume that files are in the correct format. Four additional test files are provided that include errors. Your program should not be able to load these, and should handle this in an elegant way, reporting the error to the user.

File04.wrd	Incorrect number of words
File05.wrd	Incorrect File Format
File06.wrd	Words outside of grid
File07.wrd	Incorrect overlapping words

Rubric

Criteria	Ratings					
Set up wordsearch (10%)	5.0 Pts Full marks Can setup default wordsearch. Can load wordsearches from files to setup wordsearches with horizontal, vertical and diagonal words.	3.75 Pts 1st Can setup default wordsearch. Can load wordsearches from files to setup wordsearches with horizontal and vertical words.		2.5 Pts 2:2 Can setup default wordsearch. Can load wordsearches from files to setup wordsearches with horizontal words.		1.25 Pts Fail Can setup default wordsearch.	0.0 Pts No marks No wordsearch setup.
Display wordsearch (10%)	10.0 Pts Full marks Can display wordsearch appropriately, with row and column labels highlighted yellow and found words highlighted green, and the last incorrect guess highlighted red temporarily (only the last guess should be highlighted if it was not correct). List of words included with found words also highlighted green,	7.5 Pts 1st Can display wordsearch appropriately, with row and column labels highlighted yellow and found words highlighted green. Incorrect guess highlighting may not be fully implemented. List of words included with found words also highlighted green,	6.5 Pts 2:1 Can display wordsearch appropriately, with row and column labels highlighted yellow. List of words included,	5.5 Pts 2:2 Can display wordsearch appropriately row and column labels may not be highlighted or list of words not included,	4.0 Pts Pass Can display wordsearch of the correct size, but cannot display words within the board.	2.0 Pts Fail Attempted to display wordsearch, can display rows and columns but without row or column labels, or incorrect dimensions.	0.0 Pts No marks No wordsearch displayed.
Enter coordinates (5%)	5.0 Pts Full marks Can enter rows and columns in the required format. All invalid input (letters and values out of range). All appropriate responses are given including feedback to the user stating if specific words were found.	3.75 Pts 1st Can enter rows and columns in the required format. All invalid input (letters and values out of range). All appropriate responses are given but response to invalid input is not always appropriate.		3.0 Pts 2:2 - 2:1 Can enter rows and columns in the required format. All invalid input (letters and values out of range). Some invalid input (letters or values out of range) is handled or response to invalid input is not always appropriate.		2.0 Pts Pass Can enter rows and columns but no unexpected entries (letters and values out of range) are handled.	0.0 Pts No marks Cannot enter coordinates.
Give appropriate feedback (5%)	5.0 Pts Full marks All appropriate responses are given including feedback to the user stating if specific named words were found or words removed from the word list once found or inappropriate input is accompanied by appropriate descriptive feedback. (3/3)	3.75 Pts 1st Most appropriate responses are given including feedback to the user stating if specific named words were found or words removed from the word list once found or inappropriate input is accompanied by appropriate descriptive feedback. (2/3)		2.5 Pts 2:2 Minority appropriate responses are given including feedback to the user stating if specific named words were found or words removed from the word list once found or inappropriate input is accompanied by appropriate descriptive feedback. (1/3)		2.0 Pts Pass Some appropriate feedback given but inconsistent or inappropriate.	0.0 Pts No marks No response to input.

Detect win, reset and play again option (5%)	5.0 Pts Full marks Can automatically detect wordsearch completion – offer the option to play again and reset wordsearch appropriately according to the users new inputs.		3.75 Pts 1st Can automatically detect wordsearch completion and offer the option to play again but wordsearch is not reset appropriately according to the users new inputs.		3.0 Pts 2:2-2:1 Can automatically detect wordsearch completion and give appropriate feedback – but cannot restart game.		2.0 Pts Pass Can automatically detect wordsearch completion but user is not given appropriate feedback and unable to restart game.	0.0 Pts No marks No detection of wordsearch completion
Use of Console based Menus (5%)	5.0 Pts Full marks Uses flexible, reusable menu system able to display any list of options and return a numeric choice. Code refactored well and all exceptions handled gracefully.		3.75 Pts 1st Uses flexible, reusable menu system able to display any list of options and return a numeric choice. Code refactored well or all exceptions handled gracefully.		3.25 Pts 2:1 Uses flexible, reusable menu system able to display any list of options and return a numeric choice.	2.75 Pts 2:2 Uses functional menu system able to display any list of options and return a numeric choice.	2 Pts Pass Attempt at a menu is hard coded. A functional menu presents a series of options and expects the user to enter a numeric result within a range.	0.0 Pts No marks No menu.
Saving to and Loading from a File (10%)	10.0 Pts Full marks Able to save, load and reload state accurately with all exceptions handled gracefully.	7.5 Pts 1st Able to save, load and reload state accurately with most exceptions handled gracefully.	6.5 Pts 2:1 Able to save, load and reload state with some exceptions handled gracefully.	5.5 Pts 2:2 Able to save, load and reload state.	4.5 Pts 3rd Saves or loads state only.	3.5 Pts Marginal Fail Partially saves or loads state.	2.5 Pts Fail Incomplete attempt at saving or loading state.	0.0 Pts No marks No saving or loading state.
Use of Methods (15%)	15.0 Pts Full marks All code is well refactored with no excessively long methods. Methods effectively reduced scope. Takes all opportunities to reuse code by calling methods multiple times.	11.25 Pts 1st Majority of code is well refactored with methods effectively reduced scope. Where appropriate some methods are called multiple times to effectively reuse code.	9.75 Pts 2:1 Written and called own methods to effectively reduce scope or to reuse code. However, some methods include repeated code.	8.25 Pts 2:2 Written and called own methods to effectively reduce scope or reuse code.	6.75 Pts 3rd Written own methods that do not effectively reduce scope or reuse code.	5.25 Pts Marginal Fail Uses methods provided in lectures and labs, labs or workshops.	3.75 Pts Fail Calls methods provided through the .net Framework (e.g Console.WriteLine)	0.0 Pts No marks No new methods created or called.
Use of structured data (15%)	15.0 Pts Full marks Appropriate use of structs, multidimensional arrays and enumerated types with no redundant or repeated data.		11.25 Pts 1st Appropriate use of structs, multidimensional arrays and	9.75 Pts 2:1 Appropriate use of structs, multidimensional arrays and	8.25 Pts 2:2 Use of structs and/or multidimensional	6.0 Pts Pass Data stored in one dimensional arrays.		0.0 Pts No marks No structured data used.

		enumerated types although some redundant or repeated data is included.	enumerated types although some redundant or repeated data is included.	arrays but used inappropriately.			
Exception Handling (10%)	10.0 Pts Full marks All possible exceptions handled gracefully with appropriate feedback to the user.	7.5 Pts 1st Most exceptions handled with appropriate feedback to the user.	6.0 Pts 2:2-2:1 Some exceptions handled but with a minority without appropriate feedback to the user.	4.5 Pts 3rd Some exceptions handled but a majority without appropriate feedback to the user.	3.0 Pts Fail Some exceptions handled, but result is not appropriate, and exceptions do not give appropriate feedback to the user.		0.0 Pts No marks No exception handling.
Self Commenting Code (5%)	5.0 Pts Full marks All variables and methods are appropriately named. A description of an explicit naming convention (described in a comment within the source code) is included and that convention mostly adhered to.	3.75 Pts 1st Majority of variables and methods are appropriately named. A description of an explicit naming convention (described in a comment within the source code) is included and that convention mostly adhered to.	3.0 Pts 2:2-2:1 Majority of variables and methods are appropriately named. An explicit naming convention is included (described in a comment within the source code) but that convention is more often than not adhered to.		2.0 Pts Pass Majority of variables, methods and parameters are appropriately named.	1.25 Pts Fail Minority of variables, methods and parameters are appropriately named.	0.0 Pts No marks No evidence of any sort of self commenting code. All variables, methods and parameters have non-sensical identifiers.
Use of Explicit Comments (5%)	5.0 Pts Full marks All methods include summary comments. All comments are accurate. Comments add value and do not add unnecessary clutter to code.	3.75 Pts 1st The majority of methods include summary comments. All comments are accurate. Comments add value and do not add unnecessary clutter to code.	3.0 Pts 2:2-2:1 The majority of methods include summary comments. Most comments are accurate. Most comments add value and do not add unnecessary clutter to code.		2.0 Pts Pass Some methods include summary comments. Few comments are accurate. Most comments add unnecessary clutter to code.	1.25 Pts Fail Few comments included, but add clutter and do not add value to code	0.0 Pts No marks No comments.
Use of Source Control (5%)	5.0 Pts Full marks For the entirety of the project all commits including appropriate amounts of	3.75 Pts 1st By the end of the project all commits	3.25 Pts 2:1 Several commits over time with the	2.75 Pts 2:2 Few commits with the minority	2.0 Pts Pass Very few commits with no log messages.		0.0 Pts No marks No evidence of use of source control.

	functionality and are accompanied by detailed log messages describing the update in that commit.	including appropriate amounts of functionality and are accompanied by detailed log messages describing the update in that commit.	majority including log messages and the minority of log messages that do not adequately describe the changes made in that commit.	including log messages or the majority of log messages that do not adequately describe the changes made in that commit.		