Module Learning Outcomes Assessed:

- Implement the core syntax and semantics of the programming language utilised
- Debug programs in a meaningful way using known methods and tools, diagnose and resolve syntax, logical and runtime errors
- Identify and use good principles of algorithm design to write and debug well-structured programs
- Approach and solve problems in a structured way under given constraints
- Design and implement programs in an object-oriented style and be able to explain the advantages, disadvantages and tradeoffs of choices made
- Build robust software that adheres to current conventions, that is reliable, maintainable, available and if necessary expandable and reusable

Assignment Detail

PIRATE PETE'S TREASURE HUNT GAME

At the start of the game, the user should be asked to enter the number of players in the game.

VALIDATION RULE: There has to be a minimum of 2 players and a maximum of 4 Players

Next, the user should be prompted to enter the full name and age of each player in the game (one player at a time)

VALIDATION RULE: The player has to enter their name in one line and must enter a first name and surname

VALIDATION RULE: The player's age must be 12 or over

Next, the game should (briefly) outline the rules of the game for the players. These are explained below (See "Game Rules").

Each player should be awarded 100 "Pirate Points" to start the game. Then the game should randomly give each player a number of "Dig Points"

VALIDATION RULE: The number of "Dig Points" must be a minimum of 4 and a maximum of 7.

Now subtract "Pirate Points" from each player based on their "Dig Points" x 5 – for example, if Player One gets 3 Dig Points, then subtract 3 x 5 = 15 from their "Pirate Points", so Player One now has 85 "Pirate Points" at the start of the game.

The game should then create a "Treasure Map" and display it on the screen. This will be a 10 x 10 grid of squares. The grid must be labelled so that the rows are numbered 1 to 10 and the columns are labelled A to J. This is so that the player can easily select a square.

The game will now read the location of the treasure from Pirate Pete's secret journal. This is stored in a file called "PiratePete.txt" – you should store this file in the same folder as your project so that it is easy to locate. The location of the treasure will be a row and a column on the "Map". This should NOT be shown to any player but must be stored by the game.

VALIDATION RULE: A file called "PiratePete.txt" must exist and it must contain at least two values corresponding to a map location.

VALIDATION RULE: The location must be valid (i.e. it must be a real location on the map)

VALIIDATION RULE: There must be at least one treasure location in the file, and there can be a maximum of four treasure locations. (See example files on Moodle)

The game will then ask the first player to pick a square by showing a message as follows:
"Argh..Pirate <firstname goes here>…it be your turn to dig for me treasure."

(NOTE: If at any stage, one of the Validation Rules is broken, then the game should display an appropriate error message and allow the user to re-enter data)

Then ask the user to enter a row & column (this can be done separately). This starts the game.

GAME RULES

The game <u>must follow these rules</u>:

1) Each player may only select a square that exists on the "Map"
2) A square can only be selected once (i.e. you cannot 'dig' in the same place more than once)
3) If the player finds some treasure, then add 20 "Pirate Points" to their total and mark the square on the map as having been selected (or "used"). You can decide how this is shown, but it must be clear to the user.
    a. The Player will celebrate by saying "Yo-ho-ho and a bottle of rum. I found me some pieces of eight."

4) If the player does not find any treasure, then they do not score any "Pirate Points". You will still need to show that the square has been "used"
    a. The Player will sulk by saying "Walk the plank! There be no treasure here!"

5) In all cases, subtract 1 from the Player's "Dig Points"

6) Now move to the next player and repeat the process. This will mean that you need to "re-draw" the map on the screen showing the square(s) that have been chosen so far.

7) If a Player has no "Dig Points" left, then they miss their turn. They must say "Argh, Captain, me shovel has broken!"

8) If all the treasure is found, then the game is over.

9) If no player has any "Dig Points" left, then the game is over.

When the game is over, the winner must be announced. Whichever player has the most "Pirate Points" is the winner. If there is a tie, then whoever has the most "Dig Points" left will be the winner. If both Pirate Points and Dig Points are the same, then it is a draw.

The winner (or winners, if it is a draw) will be announced by Pirate Pete, who will say "Shiver me Timbers, me hearties, sure hasn't <player name(s) goes here> won the game. Keelhaul the rest of them!"

SPECIFIC TECHNIAL REQUIREMENTS

- **You must ensure that the game follows all the Validation Rules**
- **You must use multiple user-defined classes (i.e. created by yourself!) in your final submission.**
- **Your code must be CLEARLY COMMENTED throughout to explain your code. If you do not comment your code properly, you will lose marks and could be awarded zero!**
- **You must be able to read the "Treasure Map" data from the file as specified in the description above.**
- **You are expected to follow a modular approach and create your own methods throughout your code. If all of your code is in "main" then your mark will be low.**
- **Appropriate data structure(s) must be used.**

Specific Restrictions – IMPORTANT!

- **This is NOT a graphics challenge and there are no marks for using graphics. Therefore, you are NOT permitted to us any GUI components, such as JavaSwing, JavaFX, or other.**

Marking Schedule

Description	**Weighting**
Program reads data from PiratePete.txt and creates a 10x10 board with the 'treasure' hidden on the map*	10 Marks
Program obeys the Validation Rules appropriately and provides useful error messages to the players where relevant*	20 Marks
The game rules are followed and the game can be played as intended.* There is a good user experience.	20 Marks
The program follows a modular approach with user-created methods that demonstrate good planning.*	15 Marks
The program includes multiple classes that have been created and programmed in a sensible manner*	15 Marks
Additional features or advancements have been included in the submission that go beyond the Assignment Detail provided and **make sense** for the game.*	20 Marks

(See “Suggested advancements” for possible ideas)	
Total	**100%**

*Your code MUST be clearly commented. If you do not comment your code clearly then you will lose marks in these areas and could be awarded a zero score if the commenting is very poor.

APPENDIX ONE – SAMPLE GRID

(This is to give you an idea what it might look like on screen – you're grid is unlikely to look this good !)

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

APPENDIX 2 – SUGGESTED ADVANCEMENTS

REMEMBER – These are not minimum requirements! These are only for those who might wish to aim for a very high score!

Instead of allocating "Dig Points" to a player, you could give each Player a 'shovel'. The shovel should then have a number of "Dig Points" – these reduce by one every time the player has a turn. When the shovel has no more dig points left, the player could be given the option of buying another shovel. Each shovel should have a cost that reduces "Pirate Points" (example – buy a shovel for 10 Pirate Points)

You could vary the cost of the shovel depending on how many Dig Points it has – so a shovel with 5 Dig Points would cost less than a shovel with 8 Dig Points

You could allocate each Treasure a different value. So rather than add a fixed value of 20 Pirate Points, there would be a different value for each item of treasure. You could then raise the maximum number of “treasure” from 4 to something like 8 or 10 (you don’t want to make it too easy to find treasure).

The treasure value could be assigned randomly, or from a list of possible values.