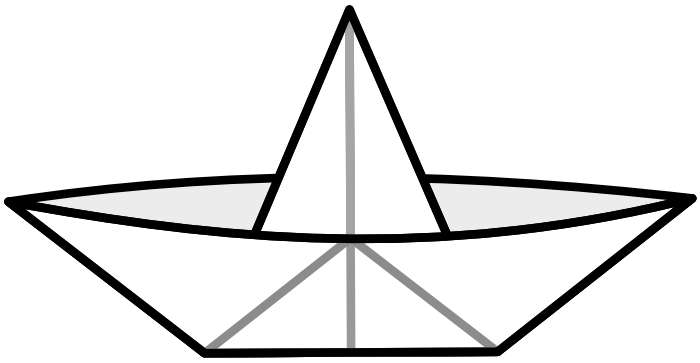

Battleboats

Zayaan Azam

Contents

1. Project Objectives

1. Create A Menu that allows the User to start a new game, load a pre-existing game, read the instructions or exit.
2. The program will validate and ensure that the inputs do correspond to a existing option and will capture the keypress, not the input itself.
3. Upon the 'New Game' option being selected, the user should be prompted to place an amount of boats (with a specified rotation) on a grid of size that is specified in the Constants.cs file.
4. The boats will be validated in order to ensure they do not get placed over each other, do not exceed the bounds of the grid and, have a valid rotation.
5. The program will, alongside the player, create a 'FleetMap' for the player that is used for computational purposes.
6. The program will then go on to generate the fleet for the computer following the same restrictions as the players boats.
7. The program will similarly create a 'FleetMap' for the computer that is used for the same purposes.
8. The program will then start the turn cycle, beginning with the player turn.
9. for each turn cycle, the player will choose a position to target.
10. said target is validated to ensure that it has not been used before.
11. this target will then be checked for, successively, a Hit > Sink > Victory.
12. The computer will then take their turn, performing the same targeting and checking cycle as the player.
13. The players grid will then be output to the player, showing the results of the computers turn.
14. this process will then be repeated until either victory is achieved for either person, where the game will end, or until escape is pressed.
15. At any point during the game loop, if the player were to press 'Escape', the game will save to disk and then display a in-game Menu
16. this menu will allow the player to either continue the game, read the instructions or save and exit the game.
17. as in the main menu, if the instructions option is selected the program will display the "Instructions.txt" file located in ./Documentation/ to the player
18. the player may then exit the instructions by pressing escape, returning to the menu.

2. Planning

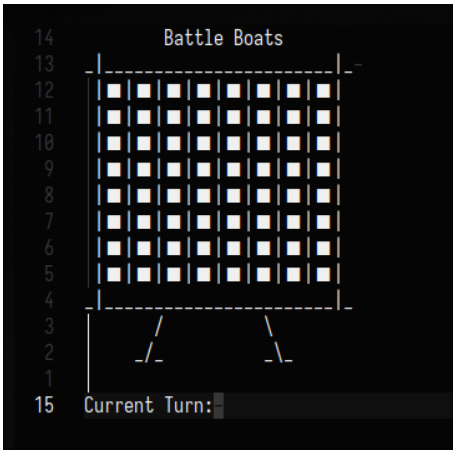

Figure 1: Initial Mockup

3. Documented Design

3.1. Function Heirarchy

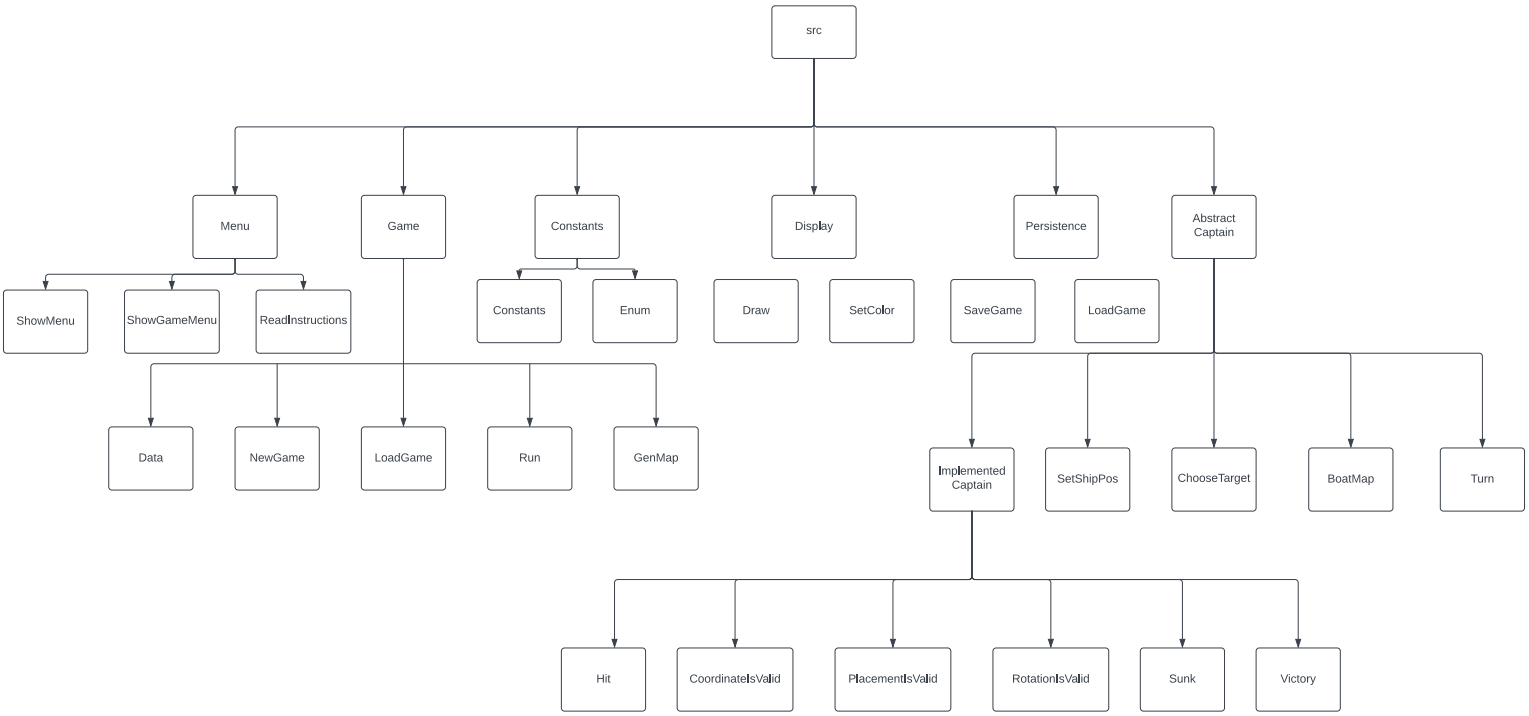

Figure 2: Software Heirarchy, In order of src > files > classes > functions. Better view can be seen in ./Documentation/Assets/heirarchy.svg

3.2. Configurable Parameters (Constants)

There are a few configurable parameters that will globally change the way the game will function. these consist of the following:

- width: the width of the playable game space
- height: the height of the playable game space
- window width: the width of the terminal, used for visual purposes, only works on windows
- window height: the height of the terminal, used for visual purposes, only works on windows
- Fleet: a list of the struct boat, used to determine whats boats are placeable for the player / computer. these can be changed and will work throughout the program, and is defined by the quantity of each boat, and the total length of said boat.

3.3. Data Structures

Tile: an enumerable, used primarily for rendering / logical purposes in order to better store the map data. Consists of the following:

- Empty1 & Empty2: logically the same, used for denoting that the cell is empty. The reasoning for 2 is to create the slight visual effect of a wave, by having a light and dark colour for the wave.
- Boat: used to denote where a boat has been placed
- Wreckage: replaces a boat, when the individual boat has been entirely hit
- Hit: Where a boat has been hit by a shot
- Miss: where a shot has missed any boats
- Using: no technical purpose, used to visually show what cell is currently being occupied by the ‘cursor’

Boat:

- a struct used in the , that stores the quantity and length of each individual boat. This is used to determine what boats can be placed and the size of them.

BoatMap:
- a struct used in the , that stores, for each boat, the seed coordinate, length, rotation, and sunk status.
- This is used primarily to determine whether a boat is either sunk or if victory has been achieved.

Data:
- a class, that holds all data that would be needed to start a new game.
- this is done for easier parameter calling and function passing, whilst also being useful for the saving and loading, due to the way newtonsoft serialises json.

Captain:
- an abstract class, used to create a form of 'template' that is then further used to create both the player and computer implementations.
- An abstract class is useful as it allows you to define abstract functions; ie functions that are just declared and not initialised. it also allows you define shared private functions, that can only be accessible by classes that inherit this abstract.

3.4. Imports
Newtonsoft.json: used for elegant serialisation and deserialisation. It allows you to convert even custom types into json compared to systems solution

3.5. Saving / Loading
Done via newtonsoft.json. instead of manually creating my own formatting for saving and loading it is done through standard json syntax. All required data is put into the Data class earlier in the program and upon serialisation, it is only that class that is saved to the file (that is specified in Constants.cs).

4. Technical Solution

4.1. Foreword
The C# Code & All Documentation can be accessed at the Git Repo and further is attached in a zip file to the teams assignment. Below listed is each file with its commented code and, a brief explanation on the purpose of the file.

4.2. Program.cs

4.3. Menu.cs

4.4. Constants.cs

4.5. Abstracts.cs

4.6. Persistence.cs

4.7. Display.cs

4.8. Game.cs