Battleship Board Game

Enrico Zammit Lonardelli

9910821

School of Physics and Astronomy

The University of Manchester

PHYS30782: Object-Oriented Programming in C++

May 2020

1 Introduction

1.1 Choice of game motivation

The board game Battleship I chose to reproduce digitally is inspired from the early 20th century pencil and paper game which later became a success board game hit in the early 1960s. This board game in particular has always had a connection with the computer world, in fact being one of the world's ever computer games being released in 1979 on the Z80 Compucolor. Since then this game has been reproduced on many digital consoles and as a standalone game on PC and is a game I have personally played many times as a child so I decided it would be a good idea to reproduce a command-line version of it in C++.

1.2 Rules

The game is played by two players. Each player has access to two grids representing their positioning of ships and the opponents' guess positions. At the start of the game each player has access to the following types ships: Once the battle begins, the first player enters a coordinate they would try to send a missile to

Name	Horizontal Length	Appearance
Carrier	5	`<-+->`
Cruiser	4	`<++>`
Destroyer	3	`<+>`
Submarine	3	`<*>`
Raft	2	`<>`

Table 1: Table listing the pieces each player must have on the board.

on the enemy's board. If the coordinate is populated by a ship, that player must say "Hit!" otherwise they must say "Miss". Depending on the outcome, the attacking player will update their enemy's guess board with the event. The other player then has the attacking turn and this is repeated until one of the players is hit on their last coordinate of their last ship.

1.3 Program Requirements

The following are the objectives set out at the beginning of the project to be completed by the end of project:

I. The program **shall**:

1. allow a human player to have their own, unique account
2. allow a human player to log in/out
3. allow a human player to start a battle from the menu
4. allow a human player to play against a computer-controlled player
5. have a computer-controlled player able to randomly select a move on their turn
6. enforce board game rules during the battle
7. recognize when the game ends and save highscores if applicable
8. have a navigable menu
9. allow a human player to view fleet through the menu
10. allow a human player to edit fleet through the menu
11. allow a human player to save fleet settings across games
12. allow a human player to view top 10 all-time game highscores through the menu
13. allow a human player to exit the game

II. The program **shall not**:

14. throw uncaught errors exiting unexpectedly
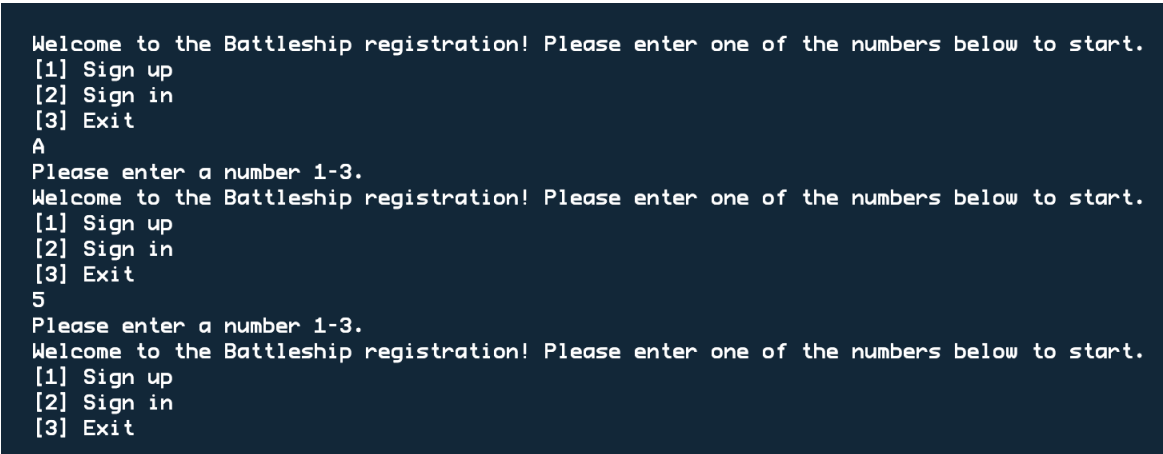15. have memory leaks

III. The program **may**:

16. allow a human player to secure their account using a hashed password
17. implement a smarter than random computer-controlled player

2 Code Design and Implementation

This program was run and tested using gcc version 6.3.0 on a x86_64-linux-gnu target running 'Debian 6.3.0-18+deb9u1'. The make file in the src/ directory includes all the options needed and running 'make' in this directory followed by moving to the build/ directory and running ./battleship will execute the program. For a fully detailed documentation (not required but makes it easier for the reader) and how to access it please refer to instructions in Appendix 2. Please note that most of the commenting explaining method motivation resides in the respective header files.
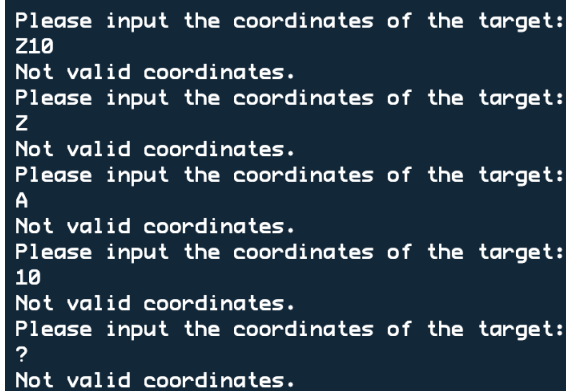
2.1 Navigation and Input Validation


```
Welcome to the Battleship registration! Please enter one of the numbers below to start.
[1] Sign up
[2] Sign in
[3] Exit
A
Please enter a number 1-3.
Welcome to the Battleship registration! Please enter one of the numbers below to start.
[1] Sign up
[2] Sign in
[3] Exit
5
Please enter a number 1-3.
Welcome to the Battleship registration! Please enter one of the numbers below to start.
[1] Sign up
[2] Sign in
[3] Exit
```

Figure 1: Screenshot of the terminal illustrating input validation on menu input.

The navigation for this program is achieved through the use of menus (numbered lists) in the command line interface. The user is required to enter a number when prompted and press the 'Enter' key. All menu input validation is done in two parts, first the std::cin failbit is checked after input to check that the input was indeed a number. Then the input is passed through a switch statement and the appropriate actions are taken. Figure 1 shows the error handling features.
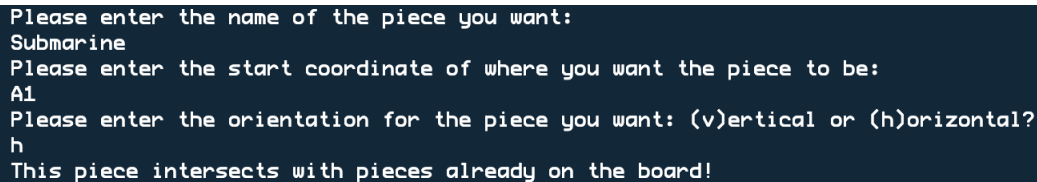
When the user is asked to input a coordinate later on in the game, this is done by adding the std::istream as friend class to the battle_ship::coordinates class. A RegEx match on this input string is done to look for letters [A-J] and 1 to 2 digits. The letter part is statically type casted to the battle_ship::x_axis enum class while the digit is cast to a std::size_t type. A check is placed in case no letters and/or digits are found. This ensures only valid letters and numbers are chosen. One further check is applied to digits to check they are within range of the board [0,10). If any of these checks do not pass, the user is notified and the method returns a falsy boolean value which the calling method of this operator should read and act accordingly. Figure 2 shows the error handling features of coordinate validation.

Figure 2: Screenshot of the terminal illustrating input validation on coordinates input.

If in the edit fleet screen the user attempts to add a piece to the board which intersects with an already existing piece the user should be notified and asked to choose another position. The same logic although can be used when a player sets a target on their turn to send a missile onto the enemy board. For this reason, the battle_ship::geometry class (seen later) was setup to check that no two pieces intersect.

Figure 3: Screenshot of the terminal illustrating input validation on a new piece input onto the board.

Lastly, Figure 4 is a top-level flowchart of how the different menus are connected and how what the user story is.
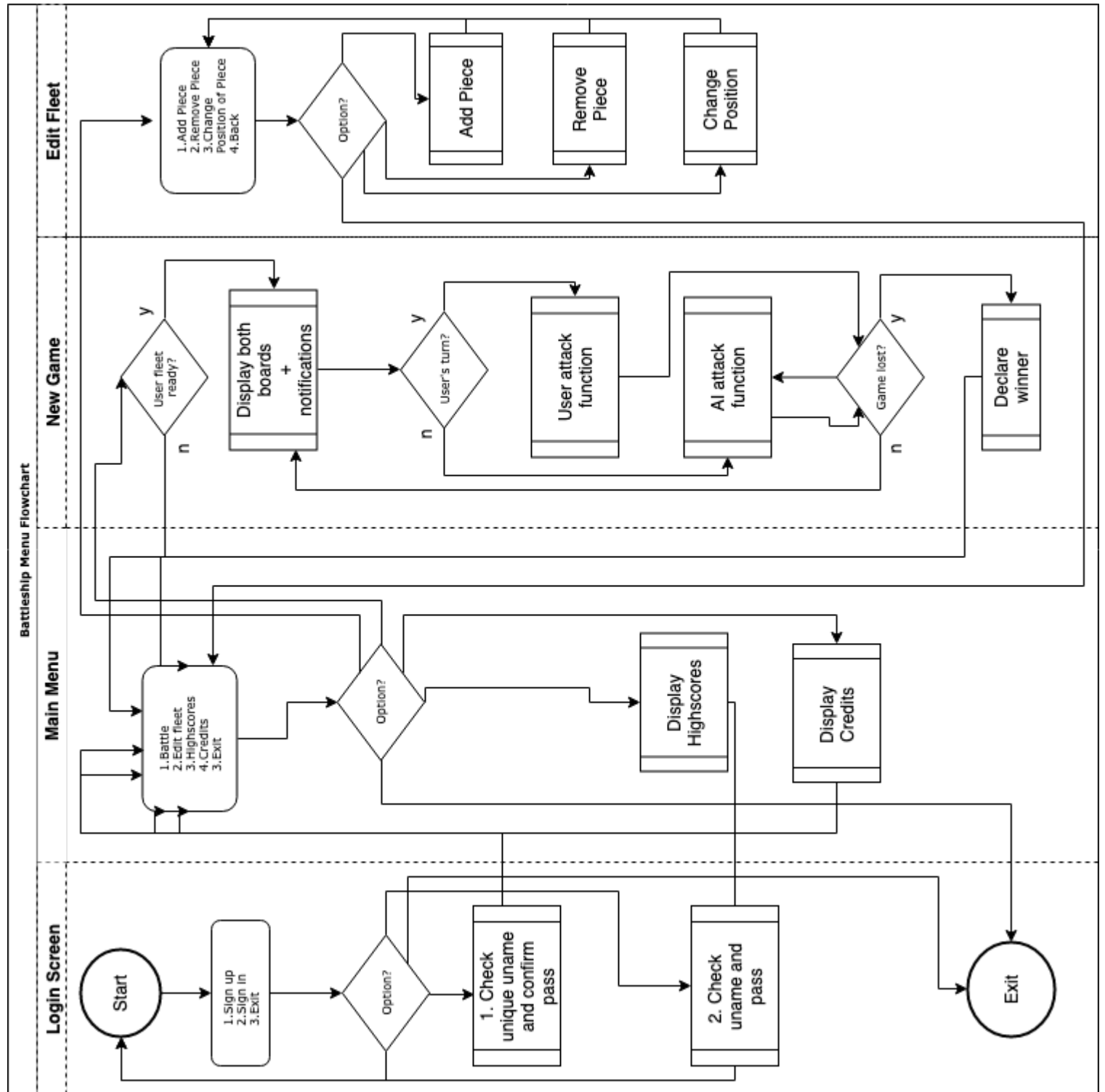
Figure 4: Flowchart detailing how different menus and 'pages' are connected.

2.2 Authentication

The battle_ship::authentication class was setup to handle signing in and up in the game. The first special feature I used here is the use of the experimental/filesystem header file. The filesystem header file is supposed to be included with C++17 however since I use gcc version 6 this still does not implement filesystem but it does have the Filesystem Technical Specification which is in experimental/filesystem. I use is std::experimental::filesystem::exists to check that the username the user wishes to register with does not exist as a savefile already. This single method call saves a lot code from having to loop through all the files and checking them one by one.

The other special feature in this class is from the std::functional header; the hash method. This method accepts a key and applies a one way map which returns an std::size_t number reprsentation for the key. Only the same key will produce the same hash map output. This is good security practice (not the best) to avoid saving passwords in plain text. Instead we save the password hash to a user profile file and whenever the user needs to login we hash their input and compare hashes. This way, the program never has to store passwords in clear text and does not explicitly have access to them.

2.3 Class Hierarchies
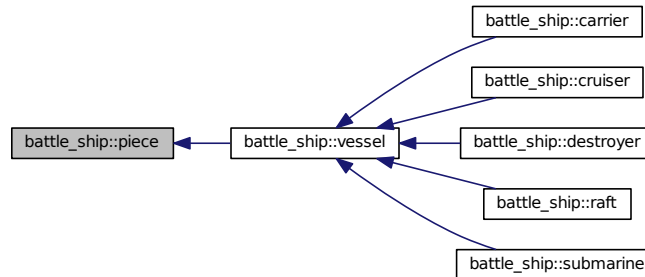
2.3.1 Pieces

Figure 5: Top level class diagram showing interface and inheritance hierarchy for the piece interface.

To make it easy to add new pieces to the game in the future and to maximise code reusability, the abstract class battle_ship::piece was created. Moreover as per C.129 of the C++ Core Guidelines the design of the inheritance hierarchy needs careful attention. I have implemented interface inheritance followed by implementation inheritance. The former is providing an interface which is detached from implementation to make it seemless when changing the actual logic of the methods for a particular implementer, without affecting the other implementers or derived classes. The latter is providing a base class which will allow derived classes to inherit its methods.

My solution is to create the battle_ship::piece class as an abstract class/interface defining all the methods its derived classes should implement. This class has a number of pure virtual methods such as attribute getters and some methods to handle modifying the pose of the piece (position and orientation). One important note is that this class does not declare any methods such as 'attack' since the thought is in the future the needs might change and there might be non-attacking pieces such as decoys added to the game. The class battle_ship::vessel was declared to implement these methods. This is a clear case of object oriented features such as inheritance explicitly as above and polymorphism with the use of dynamic linkage of the pure virtual methods defined in the base interface and implemented in the derived battle_ship::vessel class. Individual ships defined in Table 1 are then derived classes of battle_ship::vessel. They are further defined only through header files since the implementation is inherited and need only to pass different parameters to the base class

constructor. Finally, adding a new ship in the future is as easy as deifning a new header class which calls the base class' constructor with the new properties defining this ship.
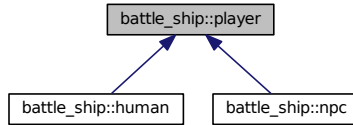
2.3.2 Player

Figure 6: Top level class diagram showing inheritance hierarchy for the player base class.

There is another class hierarchy for the concept of a player in the game. Requirement 4 in Section 1.3 requires the ability of the game to have a human player interact with a computer controlled one. For this reason I chose to create the base class battle_ship::player which does have pure virtual functions however is not solely an interface and does have private data. Through dynamic linkage and polymorphism methods such as the attack method are implemented by the derived classes battle_ship::human and battle_ship::npc (Non-player character). Through this hierarchy for example, the NPCs do not save their highscore while the humans do. They also have different attack functions where in the human's case they are asked for the target while in the NPC's case a random point is randomly selected when it is their turn (retrying only if the point had already been tried).

2.4 Manager Classes

I have implemented four classes known in the codebase as 'Manager Classes'. The purpose of these classes is to manage certain aspects such as notifications, highscores, market and screen output. Practically, these managers are used to hold data and methods frequently used by different classes through leveraging static type member data and methods.

The battle_ship::screen_manager class is used to produce the three columned output seen in the terminal so that any notifications can be shown to terminal together with the current board view. In this case it makes sense to use the static type member function since it would be unoptimized to have to instantiate this class everytime since no state needs to be maintained between calls anyways. It contains a general, repetedly used member function which needs to be accessible in different parts of the codebase. With the same reasoning the battle_ship::highscore_manager does the same with no member data but just useful methods which are used in other places of the code. This class uses an advanced C++ feature which is a generic lambda function to custom sort a vector of tuples on the second element of that tuple.

The battle_ship::market_manager is used for functionalities when editing one's fleet. This is used to encapsulate logic for commonly used methods such as selling (removing from board) or buying (adding to board) pieces. Finally, the battle_ship::notification_manager is useful to hold all current notifications in one place. All instances of the notification manager class should have the same vector of notifications and should be able to use add a notification to this vector without necessarily having to create an instance.

2.5 Board Class

The battle_ship::board class is used to keep track of each coordinate on the player's grid. It has as friend class the std::ostream to directly output to the console (via the screen manager class) and has other useful functions such as modifying a piece (with communication with the market manager class). A feature of the

output function is that it uses ANSI escape codes to display certain coordinates with colours such as an unpopulated coordinate (a blue tilde similar to a sea wave). The owner of the pieces is the board and the owner of the board is the player.

2.6 Geometry Class and Coordinates Struct

The battle_ship::coordinates struct is used everywhere in the program and defines the coordinate system used (one could change this easily thorugh this struct). Its member data consists of an enum class to define the x-axis (as this is made of letters A-J) while the y-axis is numbers (1-9).

A very important class is the battle_ship::geometry class whose public, static methods are used to calculate if one piece intersects with another on the board. This is used when adding new pieces or changing the position of pre-existing pieces in the fleet editing menu. It is also used to check if a target coordinate intersects with a piece on the board. The way this is done is by using start and end coordinates for two line segments and using simple geometric theories [1] one can calculate if the two segments intersect.

2.7 Game Class

Figure 7 shows what the terminal looks like during a battle. To the left is the player's grid, in the middle is the player's view of the enemy board showing only what has been reported given the player's attacks. To the very right is a summary of the turn-by-turn events and notifications. This is a clear use of the manager classes and their added advantage. The battle routine is the same as defined in the rules of Section 1.2.


```
Enrico's Board:                  Nemo's Board:                        Events:
     A B C D E F G H I J              A B C D E F G H I J      [Enrico] Your Carrier sends a torpedo to (F4)
    _|_____|_          _|_____|_     [Enrico] Splash...Miss!
1   | ^ ^ ^ ^ ^ ~ ~ ~ ~ ~ |    1   | ~ ~ ~ ~ ~ ~ ~ M ~ ~ |     [Nemo] The enemy's Carrier sends a torpedo to (A8)
2   | | + + * V ~ ~ ~ ~ ~ |    2   | H ~ ~ ~ ~ ~ ~ ~ ~ M |     [Nemo] The enemy has missed!
3   | + + V V ~ ~ ~ ~ ~ ~ |    3   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Enrico] Your Carrier sends a torpedo to (J2)
4   | | H ~ ~ ~ ~ ~ ~ ~ ~ |    4   | ~ ~ ~ ~ ~ M ~ ~ ~ ~ |     [Enrico] Splash...Miss!
5   | V ~ ~ ~ ~ ~ ~ ~ ~ ~ |    5   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Nemo] The enemy's Carrier sends a torpedo to (F8)
6   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |    6   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Nemo] The enemy has missed!
7   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |    7   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Enrico] Your Carrier sends a torpedo to (H1)
8   | M M ~ ~ ~ M ~ ~ ~ ~ |    8   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Enrico] Splash...Miss!
9   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |    9   | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Nemo] The enemy's Carrier sends a torpedo to (B4)
10  | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |    10  | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |     [Nemo] The enemy has hit us!
    _|_____|_          _|_____|_

Please input the coordinates of the target:
```

Figure 7: Screenshot of the game view after a few turns between a human and computer player.

2.8 Memory and Smart Pointers

The use of smart pointers was employed throughout to ensure no memory leaks occured, or at least minimise the possibility this might happen. Namely:

i. The board class uses a unique pointer to the array of strings containing the board representation at every coordinate. Moreover, it has a member vector of unique pointers to pieces held by the board.

ii. The main method uses shared pointers to players since these will then be assigned to each player as an enemy.

iii. Following this, each player holds a weak pointer to each other as an enemy. This prevents reference cycles from denying the player pointers to be dereferenced on deletion.

To check that no memory leaks were present, the program was run for all possible routines while using Valgrind (an opensource memory leak debugger) in the background and the complete log can be found in Appendix 3 detailing that **no memory leaks were found**.

3 Results

The following is a single run of the game visualised through screenshots of the most important screens and menus.

```
Welcome to the Battleship registration! Please enter one of the numbers below to start.
[1] Sign up
[2] Sign in
[3] Exit
2
Welcome back, commander. What is your username?
Enrico
Great. What is your password, sir?
abc1234
```

Figure 8: Screenshot of the first menu when launching the program for login/signup.

```
Welcome to your Battleship terminal commander! Please enter one of the numbers below to start.
[1] New Game
[2] Your Fleet
[3] Highscores
[4] Credits
[5] Exit
```

Figure 9: Screenshot of the main menu from which different routines are launched.

```
These are the top 10 commanders of all time!
Commander              Turns
Professor                20
Enrico                   23
Leela.T                  25
Philip.J.F                30
```

Figure 10: Screenshot of the all-time highscores panel.

```
Enter one of the numbers below to execute that order.
[1] Add piece
[2] Remove Piece
[3] Change position of piece
[4] To main menu
```

Figure 11: Screenshot of the fleet editing menu.

```
These are the available pieces to be added. Please select one:
Destroyer
Please enter the name of the piece you want:
Destroyer
Please enter the start coordinate of where you want the piece to be:
B1
Please enter the orientation for the piece you want: (v)ertical or (h)orizontal?
v
Transaction successful.This is your current configuration.
     A B C D E F G H I J
    __|_____|
1  | ~ ^ ~ ^ ^ ~ ~ ~ ~ ~ |
2  | ^ + ~ * V ~ ~ ~ ~ ~ |
3  | | V ~ V ~ ~ ~ ~ ~ ~ |
4  | + ~ ~ ~ ~ ~ ~ ~ ~ ~ |
5  | | ~ ~ ~ ~ ~ ~ ~ ~ ~ |
6  | V ~ ~ ~ ~ ~ ^ ~ ~ ~ |
7  | ~ ~ ~ ~ ~ ~ + ~ ~ ~ |
8  | ~ ~ ~ ~ ~ + ~ ~ ~ |
9  | ~ ~ ~ ~ ~ ~ V ~ ~ ~ |
10 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
    __|_____|
```

Figure 12: Screenshot of adding pieces to fleet, if not all the pieces have been added already.
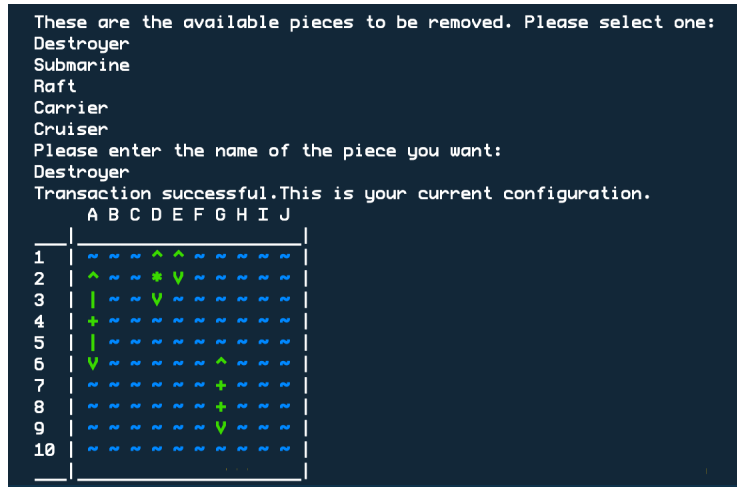
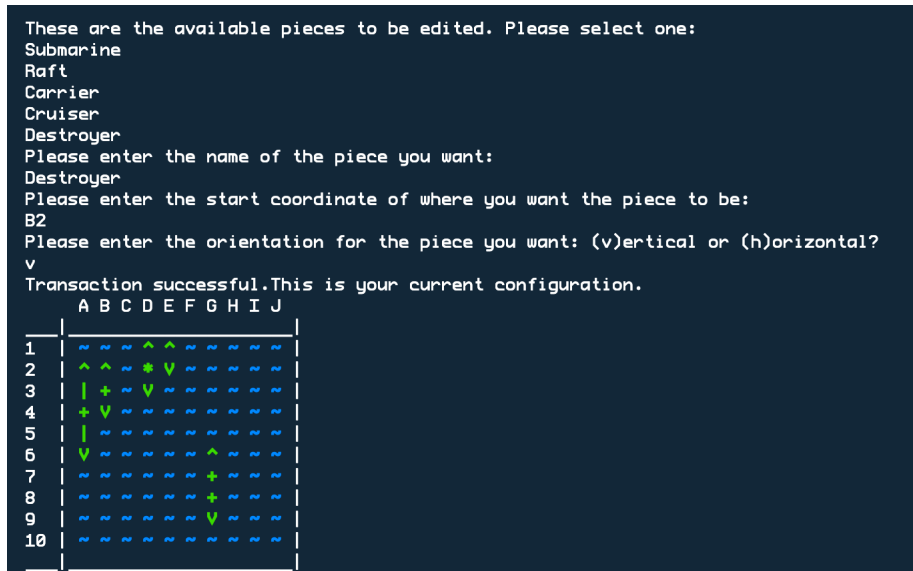Figure 13: Screenshot of deleting a piece from the fleet, if the fleet is not empty.

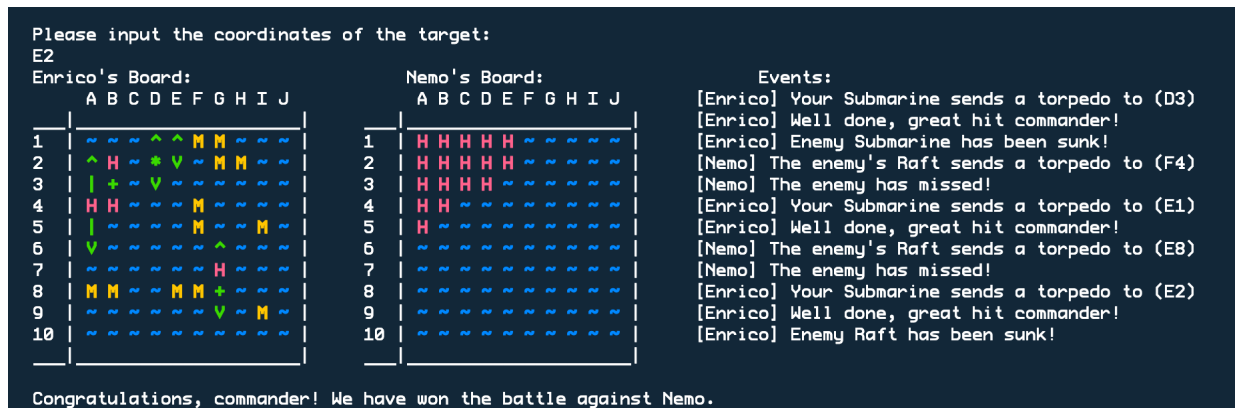Figure 14: Screenshot of changing position of a piece in the fleet.

Figure 15: Screenshot of winning the game against the NPC.

4   Conclusions and Extensions

All required functionality set out at the beginning in Section 1.3 was completed successfully. The whole program was tested for input validation and sanitization together with memory checks to make sure no leaks occur. Finally, object oriented features such as inheritence, polymorphism, encapsulation and abstraction were all implemented. Advanced C++ features such as the use of generic lambda functions, hashmaps, tuples and all three types of smart pointers were used.

As possible extensions, even though currently every piece has the same cost i.e 1 and the budget each player has is the number of pieces, one could very easily change this in the future by constraining the budget and maybe adding more vessels with different costs opening a really big game element such as is choosing your fleet correctly. Moreover, since the game saves fleet state after closing, creating one's fleet wisely has even more value. Finally, implementing a smarter NPC strategy than random selection would be interesting possibly with different levels of opponent difficulty too.

<div align="right">2410 words</div>

References

[1]   P. Prosser, "Algorithms and data structures course notes," Geometric Algorithms Slides.

Appendix 1: Impact of COVID19 on the work completed and report

4.1   Time lost travelling home

On the 16th of March I caught the first airplane home (Malta) I could find, due to news that airports back home were imminently closing. As a matter of fact a week later they did close permanently and so have the UK soon after that. The time I did lose was not explicitly in travelling home although that was substaintial (around 10 hours all together) it was more since I left almost imminently I left a lot of resources back in Manchester (such as books, documents) and it has been really painful trying to find alternatives to them.

4.2   Lack of access to computers and the internet

I am very lucky to have no problem on this end since I have a laptop I could take with me and I have a stable internet connection. In fact I attended all the lectures on BlackBoard Collaborate that occured for this and other modules.

4.3   Limitation in the outcome

I honestly do not think it has affected me in any way towards this particular module. The lecturer and lab assistants all have been very helpful online whenever I needed and learning resources, perhaps because this is a programming module, were fairly easy to find online.

Appendix 2: Instructions for Opening Optional Full Documentation

Attached to the .cpp and .h files should be a directory called html. This directory is not required to run the program but rather is a fully compiled documentation of the whole code. It is a nice, UI friendly way to check the codebase. The way to open it is to enter the directory and open the file 'index.html' in a browser.

Appendix 3: Log of Valgrind Memory Check

```
==64058== Memcheck, a memory error detector
==64058== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==64058== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==64058== Command:  ./battleship
==64058== Parent PID: 44655
==64058==
--64058--
--64058-- Valgrind options:
--64058--    --leak-check=full
--64058--    --show-leak-kinds=all
--64058--    --track-origins=yes
--64058--    --verbose
--64058--    --log-file=valgrind-out.txt
--64058-- Contents of /proc/version:
--64058--   Linux version 4.19.76-linuxkit (root@d203b39a3d78) (gcc version 8.3.0 (Alpine 8.3.0))
#1 SMP Thu Oct 17 19:31:58 UTC 2019
--64058--
--64058-- Arch and hwcaps:  AMD64, LittleEndian, amd64-cx16-lzcnt-sse3-avx-avx2-bmi
--64058-- Page sizes:  currently 4096, max supported 4096
--64058-- Valgrind library directory:  /usr/lib/valgrind
--64058-- Reading syms from /workspaces/OO_CPP/Final_Project/build/battleship
--64058-- Reading syms from /lib/x86_64-linux-gnu/ld-2.24.so
--64058--   Considering /usr/lib/debug/.build-id/60/6df9c355103e82140d513bc7a25a635591c153.debug
..
--64058--   .. build-id is valid
--64058-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--64058--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--64058--   .. CRC mismatch (computed db5b2ec5 wanted 0eae776b)
--64058--   Considering /usr/lib/debug/usr/lib/valgrind/memcheck-amd64-linux ..
--64058--   .. CRC is valid
--64058--    object doesn't have a dynamic symbol table
--64058-- Scheduler:  using generic scheduler lock implementation.
--64058-- Reading suppressions file:  /usr/lib/valgrind/default.supp
==64058== embedded gdbserver:  reading from /tmp/vgdb-pipe-from-vgdb-to-64058-by-???-on-8c2541b3ea
==64058== embedded gdbserver:  writing to   /tmp/vgdb-pipe-to-vgdb-from-64058-by-???-on-8c2541b3ea
==64058== embedded gdbserver:  shared mem   /tmp/vgdb-pipe-shared-mem-vgdb-64058-by-???-on-8c2541b3
==64058==
==64058== TO CONTROL THIS PROCESS USING vgdb (which you probably
==64058== don't want to do, unless you know exactly what you're doing,
==64058== or are doing some strange experiment):
==64058==   /usr/lib/valgrind/../../bin/vgdb --pid=64058 ...command...
==64058==
==64058== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==64058==   /path/to/gdb ./battleship
==64058== and then give GDB the following command
==64058==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=64058
==64058== --pid is optional if only one valgrind process is running
==64058==
--64058-- REDIR: 0x401aec0 (ld-linux-x86-64.so.2:strlen) redirected to 0x3809de81 (vgPlain_amd64_li
--64058-- REDIR: 0x4019770 (ld-linux-x86-64.so.2:index) redirected to 0x3809de9b (vgPlain_amd64_lin
```

```
--64058-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--64058--    Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--64058--     .. CRC mismatch (computed 74a069fa wanted 84d99202)
--64058--    Considering /usr/lib/debug/usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--64058--     .. CRC is valid
--64058-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
--64058--    Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so ..
--64058--     .. CRC mismatch (computed 88f2547e wanted 8a7a4459)
--64058--    Considering /usr/lib/debug/usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so ..
--64058--     .. CRC is valid
==64058== WARNING: new redirection conflicts with existing -- ignoring it
--64058--     old: 0x0401aec0 (strlen                ) R-> (0000.0) 0x3809de81 vgPlain_amd64_linux_R
--64058--     new: 0x0401aec0 (strlen                ) R-> (2007.0) 0x04c2ee60 strlen
--64058-- REDIR: 0x4019990 (ld-linux-x86-64.so.2:strcmp) redirected to 0x4c2ff60 (strcmp)
--64058-- REDIR: 0x401b9d0 (ld-linux-x86-64.so.2:mempcpy) redirected to 0x4c33330 (mempcpy)
--64058-- Reading syms from /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.22
--64058--    object doesn't have a symbol table
--64058-- Reading syms from /lib/x86_64-linux-gnu/libm-2.24.so
--64058--    Considering /usr/lib/debug/.build-id/4e/49714c557ce0472c798f39365ca10f9c0e1933.debug
..
--64058--     .. build-id is valid
--64058-- Reading syms from /lib/x86_64-linux-gnu/libgcc_s.so.1
--64058--    object doesn't have a symbol table
--64058-- Reading syms from /lib/x86_64-linux-gnu/libc-2.24.so
--64058--    Considering /usr/lib/debug/.build-id/77/5143e680ff0cd4cd51cce1ce8ca216e635a1d6.debug
..
--64058--     .. build-id is valid
--64058-- REDIR: 0x5759710 (libc.so.6:strcasecmp) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x57551b0 (libc.so.6:strcspn) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x575ba00 (libc.so.6:strncasecmp) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x5757620 (libc.so.6:strpbrk) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x57579b0 (libc.so.6:strspn) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x5758d80 (libc.so.6:memmove) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x5757330 (libc.so.6:rindex) redirected to 0x4c2e7f0 (rindex)
--64058-- REDIR: 0x574ff10 (libc.so.6:malloc) redirected to 0x4c2bb40 (malloc)
--64058-- REDIR: 0x5755650 (libc.so.6:strlen) redirected to 0x4c2eda0 (strlen)
--64058-- REDIR: 0x5758980 (libc.so.6:__GI_memcmp) redirected to 0x4c31b40 (__GI_memcmp)
--64058-- REDIR: 0x5753c00 (libc.so.6:strcmp) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x5764520 (libc.so.6:__strcmp_sse2_unaligned) redirected to 0x4c2fe20 (strcmp)
--64058-- REDIR: 0x4ec7790 (libstdc++.so.6:operator new(unsigned long)) redirected to 0x4c2c1b0
(operator new(unsigned long))
--64058-- REDIR: 0x575e100 (libc.so.6:memcpy@@GLIBC_2.14) redirected to
0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x57fe020 (libc.so.6:__memcpy_avx_unaligned_erms)
redirected to 0x4c325c0 (memmove)
--64058-- REDIR: 0x5758940 (libc.so.6:bcmp) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x5818f80 (libc.so.6:__memcmp_sse4_1) redirected to 0x4c31ca0 (__memcmp_sse4_1)
--64058-- REDIR: 0x5758e70 (libc.so.6:__GI_mempcpy) redirected to 0x4c33060 (__GI_mempcpy)
--64058-- REDIR: 0x4ec58c0 (libstdc++.so.6:operator delete(void*)) redirected to 0x4c2d270
(operator delete(void*))
--64058-- REDIR: 0x5757f50 (libc.so.6:__GI_strstr) redirected to 0x4c33590 (__strstr_sse2)
--64058-- REDIR: 0x4ec7850 (libstdc++.so.6:operator new[](unsigned long)) redirected
```

to 0x4c2c8d0 (operator new[](unsigned long))
--64058-- REDIR: 0x4ec58f0 (libstdc++.so.6:operator delete[](void*)) redirected to 0x4c2d770
(operator delete[](void*))
--64058-- REDIR: 0x5750510 (libc.so.6:free) redirected to 0x4c2cd70 (free)
--64058-- REDIR: 0x57539b0 (libc.so.6:index) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x57539e0 (libc.so.6:__GI_strchr) redirected to 0x4c2e950 (__GI_strchr)
--64058-- REDIR: 0x57557f0 (libc.so.6:strnlen) redirected to 0x4c2ed40 (strnlen)
--64058-- REDIR: 0x5758e90 (libc.so.6:memcpy@GLIBC_2.2.5) redirected
to 0x4c300c0 (memcpy@GLIBC_2.2.5)
--64058-- REDIR: 0x5759230 (libc.so.6:memset) redirected to 0x4a26740 (_vgnU_ifunc_wrapper)
--64058-- REDIR: 0x57fe470 (libc.so.6:__memset_avx2_unaligned_erms) redirected to 0x4c324c0
(memset)
--64058-- REDIR: 0x57585f0 (libc.so.6:memchr) redirected to 0x4c30000 (memchr)
==64058==
==64058== HEAP SUMMARY:
==64058==   in use at exit: 0 bytes in 0 blocks
==64058==   total heap usage: 39,921 allocs, 39,921 frees, 1,347,977 bytes allocated
==64058==
==64058== All heap blocks were freed -- no leaks are possible
==64058==
==64058== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==64058== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)