

Assessment 1 - ZE - 1908686

by Zack Evans

Submission date: 07-Dec-2021 04:21PM (UTC+0000)

Submission ID: 165291860

File name: 18253_Zack_Evans_Assessment_1_-_ZE_-_1908686_382345_527904046.docx (302.74K)

Word count: 3137

Character count: 14865

Advanced Software Development

Level 5 Assessment – 1

Table of Contents

Table of Contents.....	2
Introduction	3
OOP Concepts	4
Classes & Objects	4
Example	4
Encapsulation	7
Project Example	8
Abstraction.....	10
Project Example	11
Inheritance	12
Project Example	13
Polymorphism	13
Project Example	14
Battleship	15
Development overview	15
Testing.....	16
Main Menu.....	16
Player names.....	18
Ship placements	19
Player guesses	21
Grid check	23
References	26

Introduction

The purpose of this assessment report is to provide the concepts of Object Orientated Programming and the overall importance of OOP. This report will illustrate the main concepts and provide examples from the code produced to make the game.

A written table of tests will be supplied and evidence of all user handled data will be done to make the process of playing the game run as smooth as possible.

OOP Concepts

OOP or Object-oriented programming is based on the concept that relies on objects. It's used to create simple and reusable classes. These can be used within a program to reduce repetitive code while also allowing the creation of individual instances of the objects.

Classes & Objects

The classes or objects are templates that can hold values and functions that can be edited and ran once created. This can be references throughout a program to easily manipulate data. An example of this would be an animal. An animal has multiple attributes [1]:

- Colour
- Noise it makes
- Carnivore?

While also having functions like:

- Noise
- Eats

With this we can create multiple animals with different attributes. With this depending on the attributes the functions can be made to display different values. Such as if we made an object with:

- Yellow
- Rawr
- Carnivore? (True)

Then the noise it would make would be "Rawr". While it eats "Meat". If we created another object that's:

- Brown
- Squeak
- Carnivore? (False)

Then the noise it would make would be "Squeak". While it would eat "Leaves and Nuts". The functionality within the class can be edited to the presence of the object it represents. Meaning it could output a static text that's fixed no matter what attributes you assign it. While you can create dynamic functions that output different values depending on the attributes set.

Example

Within the Battleship project, a Ship class was made to keep the consistency of different ships the same. A base constructor was made to store the name of the ship as well as the length. Once set the values can be called upon via the get methods.

The best example where the ship object is used the most is within the "editGrid" function within the game controller class. It takes in both the players grid as well as the ship that needs to be placed. It retrieves the coordinates, direction and length of the ship. And edits the player grid depending on the direction, length and x & y positions.

```
#pragma once
#include <string>
using namespace std;

class Ship
{
private:
    string name;
    int length;
    int xy[2];
    int direction;
public:
    Ship();
    Ship(string name, int length);
    string getName();
    int getLength();
    void setDirection(int direction);
    int getDirection();
    void setCoordinates(int x, int y);
    int* getCoordinates();
};
```

```
#include "Ship.h"

Ship::Ship()
{
}

Ship::Ship(string name, int length) {
    this->name = name;
    this->length = length;
}

string Ship::getName()
{
    return this->name;
}

int Ship::getLength()
{
    return this->length;
}

void Ship::setDirection(int direction)
{
    this->direction = direction;
}

int Ship::getDirection()
{
    return this->direction;
}

void Ship::setCoordinates(int x, int y)
{
    this->xy[0] = x;
    this->xy[1] = y;
}

int* Ship::getCoordinates()
{
    return this->xy;
}
```

```

void Game::editGrid(char playerGrid[10][10], Ship ship) {
    int* xy = ship.getCoordinates();
    int coY = xy[0] - 1,
        coX = xy[1] - 1;

    switch (ship.getDirection()) {
    case 1:
        // Up Direction
        for (int x = 0; x < ship.getLength(); x++) {
            playerGrid[coX - x][coY] = '-';
        }
        break;
    case 2:
        // Right Direction
        for (int x = 0; x < ship.getLength(); x++) {
            playerGrid[coX][coY + x] = '-';
        }
        break;
    case 3:
        // Down Direction
        for (int x = 0; x < ship.getLength(); x++) {
            playerGrid[coX + x][coY] = '-';
        }
        break;
    case 4:
        // Left Direction
        for (int x = 0; x < ship.getLength(); x++) {
            playerGrid[coX][coY - x] = '-';
        }
        break;
    }
}

```

Encapsulation

This is defining the idea of combining data with methods to keep information hidden. The general idea of encapsulation is simple. Within the class, if you have an attribute that is not visible from the outside (Cannot be directly accessed or referenced). Then this can be used with methods to provide read or write privileges. Overall, you can hide specific data within an object while also being able to provide control to the internal attributes [2].

Within OOP, methods called “getters” and “setters” are used to control the hidden attributes. As the name indicates the getter method retrieves the data from the attribute while the setter method can be used to change the hidden attribute.

For an example, if a class had the attributes:

- Age
- Height

Zack Evans
1908686

Then we can set getters and setters of:

- `getAge`
- `getHeight`
- `setAge`
- `setHeight`

This can then be used to get and set data that's being parsed through the object. Other methods can be used to also manipulate hidden attributes. An example of this would be if we don't want the height in centimetres, we can create a function to return feet and inches instead. So creating another method called `"getHeightInFeet"`, would return the needed format of the original data.

Overall, encapsulation can be used to set and get raw data. Or it can be used to manipulate formatted data to the raw format and then stored, or vice versa.

[Project Example](#)

Yet again if we use the ship class due to encapsulation being used. We can see that we have several values that are private. Meaning that they cannot be directly accessed. Some attributes can be referenced and changed via getters and setters. While others can only be read. With this we can see overall there are some attributes with read and write privileges:

- `Direction`
- `XY`

While others like:

- `Name`
- `Length`

Can only be read once the constructor has been used with the data that's been parsed into it.

```
#pragma once
#include <string>
using namespace std;

class Ship
{
private:
    string name;
    int length;
    int xy[2];
    int direction;
public:
    Ship();
    Ship(string name, int length);
    string getName();
    int getLength();
    void setDirection(int direction);
    int getDirection();
    void setCoordinates(int x, int y);
    int* getCoordinates();
};
```

```

#pragma once
#include <string>
using namespace std;

class Ship
{
private:
    string name;
    int length;
    int xy[2];
    int direction;
public:
    Ship();
    Ship(string name, int length);
    string getName();
    int getLength();
    void setDirection(int direction);
    int getDirection();
    void setCoordinates(int x, int y);
    int* getCoordinates();
};

```

Abstraction

The main goal of abstraction is to handle a complex process and hide its unnecessary details from the user. A real-world example of this would be a user who uses a kettle. The user only needs to be able to see the water level, fill up the kettle and turn it on. The user doesn't need to know the temperature of the water or how much power is needed to heat up the water [3].

In OOP, abstraction allows the developer to easily implement complex functionality while also being able to setup the class to only show relevant methods and attributes that outside variables need to interact with it. This means not only can the developer implement a complex process, but the developer can also implement a new process that contains the already existing process. Overall, abstraction allows the use of hiding and showing methods like attributes to allow the object to be referenced for start processes while also hiding the middle and end processes that don't need to be shown.

A good example of this would be to create a class called Kettle. This would contain only two public functions:

- Turn kettle on
- Fill kettle

While having multiple hidden functions:

- Heat water
- Check water temperature
- Turn kettle off

Zack Evans
1908686

The process would look like:

1. Fill up kettle (Public) – Would allow the user to increase a hidden attribute that stores water contents
2. Turn on kettle (Public) – Allows the user to turn on the kettle to heat up water
3. Heats up water (Private) – This heats up the water
4. Check water temperature (Private) – Checks to see if the water has reached boiling point
5. Repeat steps 3 & 4
6. Turn of kettle

While the user only needs to handle two functions. The overall process uses several steps and methods to run the entire process. This is a good example of showing a complex process without needing to show the complex workings that are behind the scenes.

Project Example

The best project example would be the game controller class. This class handles all the game functionality while also allowing the user to start the game, repeat the game or quit. In the Battleship game, the class can only be referenced for start game. Which is the only public function. While all the other complex methods are hidden in the background since it doesn't need to be referenced externally.

```
#pragma once
#include "Common.h"
#include "Ship.h"
#include "Player.h"
#include "Menu.h"

class Game
{
private:
    int MAX_COLS;
    int MAX_ROWS;
    Common common;
    Menu menu;
    Ship ships[2];
    //
    void grid(char a[10][10], bool show);
    Player placeShips(Player player);
    void editGrid(char playerGrid[10][10], Ship ship);
    bool checkGrid(char playerGrid[10][10], Ship ship);
    Player playerGuess(Player player, int xy[2]);
    bool checkGame(Player player);
public:
    Game();
    void start();
};
```

```

void Game::start() {
    system("CLS");

    Player player1(common.getInput("Player 1 name:"));
    Player player2(common.getInput("Player 2 name:"));

    player1 = this->placeShips(player1);
    system("CLS");
    player2 = this->placeShips(player2);
    system("CLS");

    Player players[2] = { player1, player2 };

    while (true) {
        int num = 0;
        for (Player player : players) {
            system("CLS");

            cout << "Player " << players[num == 0 ? 1 : 0].getName() << " turn to
guess." << endl;

            this->grid(player.grid, false);

            int xy[2];
            bool valid = false;
            while (!valid) {
                int x = common.getNumber("X:", 1, 10);
                int y = common.getNumber("Y:", 1, 10);
                xy[0] = x; xy[1] = y;

                if (player.grid[y - 1][x - 1] != 'X' &&
                    player.grid[y - 1][x - 1] != '#') valid = true;
                else {
                    cout << "Coordinate already guessed. Try again." << endl;
                    cin.get();
                }
            }

            players[num] = this->playerGuess(player, xy);

            if (this->checkGame(players[num == 0 ? 0 : 1])) {
                system("CLS");
                EndMenu endMenu(players[num == 0 ? 1 : 0].getName());
                endMenu.print();
                cin.get();
                return;
            }

            num++;
        }
        cin.get();
    }
}

```

Inheritance

Inheritance is a feature that's developed into classes that allows developers to create a hierarchy that shares both attributes and methods. A class that has been derived from are called "superclass" or "parent class". The derived class is called "subclass" or "child class" [4].

When defining a subclass, the keyword “extends” is used to extend from a parent class. This allows the class to share the parent’s attributes and methods. An example of inheritance would be to have a base class called “Animal”. This would hold basic attributes and methods that define the animal. Then a derived class called “Cat” or “Dog” could be created from the animal class. This allows us to add additional attributes and methods.

Project Example

The project example would be the Menu and EndMenu class. The Menu class is the parent and the EndMenu class is the child.

```
#pragma once

#include "Common.h"
using namespace std;

class Menu
{
private:
    Common common;
    void banner();
    void options();
public:
    void print();
    void directions();
};

#pragma once

#include "Menu.h"
using namespace std;

class EndMenu : public Menu
{
private:
    string playerName;
public:
    EndMenu(string player);
    void print();
    void banner();
};
```

Polymorphism

Like inheritance, polymorphism is a feature that’s build within classes that grants the ability of an object to take on many forms. Inheritance allows a develop to inherit attributes and methods while also allowing them to add new attributes and methods. While polymorphism allows the developer to alter existing attributes that derive from a parent class. This means constructors, attributes and methods can be altered depending on the needs of the child class.

This allows a developer to create an animal class that makes the sound “General Animal Noise”. This then can be used to create a child class called dog. The developer can edit the already existing method that displays the animal noise, the attribute that stores the noise. This can then display “Bark” instead of the general animal noise. With this, developers can use this in many use cases such

as making a generic person class and then deriving that into a student class and lecturer class that both store additional values as well as have altered classes depending on the circumstances.

Project Example

Due to the only example of inheritance being the Menu and EndMenu classes. We will be overwriting methods within the EndMenu to display a different output when a player wins.

```
#include "EndMenu.h"

EndMenu::EndMenu(string player)
{
    this->playerName = player;
}

void EndMenu::print() {
    this->banner();
    cout << endl;
}

void EndMenu::banner() {
    cout << "#####" << endl;
    cout << endl;
    cout << "    Winner: " << this->playerName << endl;
    cout << endl;
    cout << "#####" << endl;
}
```



Battleship

For the assessment, the objective was to create or recreate a game in C++. The game chosen for the assessment was Battleship, due to straight forward rules but some complexity within developing the game.

Battleship is a strategy type guessing game where two players play on a grid-based board. At the start of the game the players place the ship markers on the board which is concealed from the other player. Players alternate turns where they guess where the other players ships are. The one and only objective of the game is to destroy the other player's entire fleet.

Here is the link to the video demonstration:

<https://www.youtube.com/watch?v=rdbiDpl5bwo>

Development overview

Before developing the game, multiple aspects of the game are needed to be analysed to get a clear view on what needs to be done and setup before implementing the functionality of the game. From the development view, the main features would be:

- Menu
- Player
- Game Controller

The menu would handle user options such as start game and quit game, while the player would store all the player related like the player grid and ship placements. And lastly, the game controller would handle the player data depending on the user's inputs.

The menu will be a simple feature that handles the users requests to start or quit the game. So making a simple class the handle just graphical outputs can be used without the need of storing anything.

The player would be handled as a class. This will be used to store the players name and grid. So having functions to set the values while also being able to retrieve them.

The game controller will need to do multiple things:

- Create two players
- Allow players to setup their grids
- Allow users to guess the opponents grid
- Check for validation
- Check if the player has won

At the start of the game, two player objects will need to be created allowing the user to enter their identifiers so that the game to call them to take their turn. Next the controller would need to handle both players and allow them to edit their grids with the ship pieces. Once the players have placed all five ships, the next phase starts.

The players will need to be able to guess the opponents ship placements. Meaning that the function that shows the opponents board will need a hide condition to hide the placements while also showing the hits and misses that they have guessed.

For validation, first the game needs to check if the coordinates are valid. If not, then they will need to be re-entered. If the user is placing a ship the length will need to be taken account and the

Zack Evans
1908686

placement. Meaning that if the player places the ship at 1,1 (Top left of the grid) and tries to place it facing upwards (One of the ships markers would be out of bounds). The game will need to warn what the user is trying to do and let the user place the ship again in another location and direction. When guessing, the coordinate validation will take place while also another check will happen to check if the guess placement has already been guessed. If the game detects that a hit or a miss and already been placed in the grid location, then the user will need to retake their guess.

Lastly, when the game detects a guess from a player. The game will need to run a check to see if all ship markers have been guessed. If so, then the player has won. If not, the game will continue. If a player wins a new menu will need to be displayed showing the winners name and replay and quit functionality. For this inheritance and polymorphism can be used to create a new menu class to create the end menu.

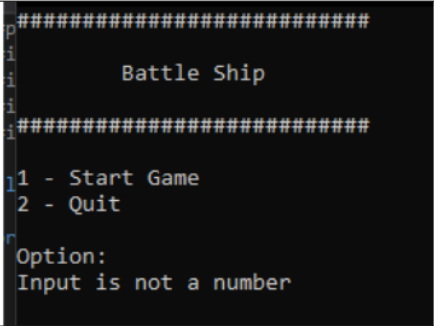
Testing

When testing, the main goals was to create a game where user error is kept to a minimum. So testing on all user inputs where done as well as on the dynamic battleship grid.

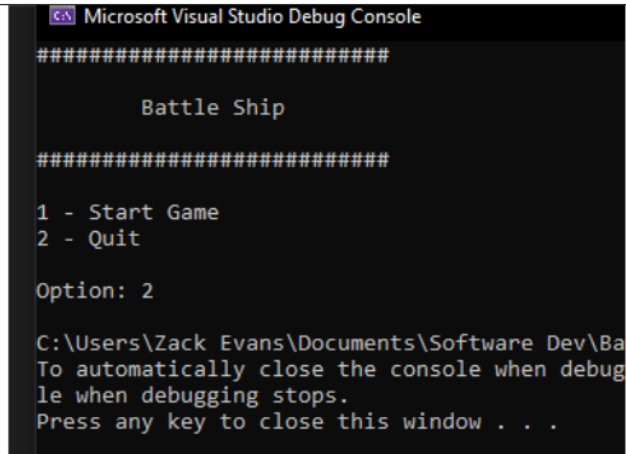
Main Menu

When testing the main menu that only uses numbers to guide the user. Consideration had to be taken. Multiple scenarios where created to test every input that can be done. These are:

- Blank entry
- Only letters
- Only numbers
- Negative numbers
- Mixed characters
- Characters with spaces
- Valid input

Test	Description	Evidence	Outcome
1	This test checks the outcome on what happens when a blank input is used		Due to the input not being a number. The default exception is sent

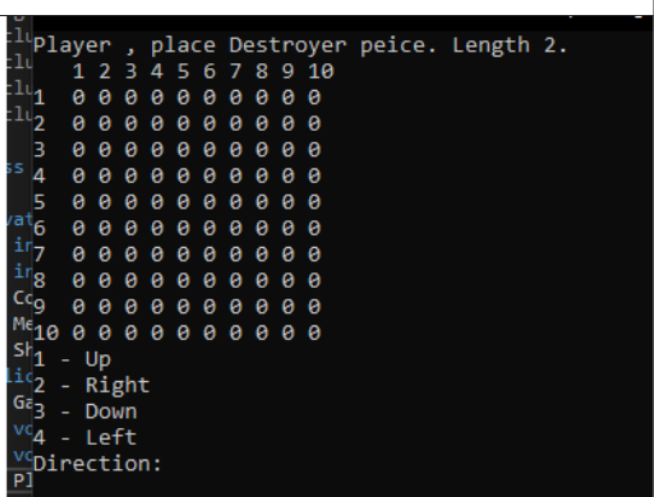
2	What happens when an invalid input is entered	<pre>##### #p##### #i #i Battle Ship #i #i##### #i cl1 - Start Game { 2 - Quit pr Option: abc Input is not a number _</pre>	Same as test 1
3	What happens when an invalid number is used	<pre>##### #p##### #i #i Battle Ship #i #i##### #i cl1 - Start Game { 2 - Quit pr Option: 3_ : : ##### #p##### #i #i Battle Ship #i #i##### #i cl1 - Start Game { 2 - Quit pr Option:</pre>	The process runs through the switch statement. But due to no clauses are matching the input. It refreshes the menu waiting for the user to insert a valid input.
4	What happens when a user starts the game	<pre>##### #p##### #i #i Battle Ship #i #i##### #i cl1 - Start Game { 2 - Quit pr Option: Player 1 name:</pre>	It moves to the game start method

5	What happens if the user quits	 <pre> Microsoft Visual Studio Debug Console ##### Battle Ship ##### 1 - Start Game 2 - Quit Option: 2 C:\Users\Zack Evans\Documents\Software Dev\Ba To automatically close the console when debug le when debugging stops. Press any key to close this window . . . </pre>	The application exists with the code of 0
---	--------------------------------	---	---

Player names

Just like the menu test. An input needs to be tested to see if the correct text is showing. When the user inserts any type of characters. It should save and display. The tests done would be:

- Blank entry
- Any character

Test	Description	Evidence	Outcome
1	When a user leaves the entry blank	 <pre> player , place Destroyer peice. Length 2. 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 Sh 1 - Up 2 - Right 3 - Down 4 - Left vc Direction: Pl </pre>	This leaves a blank username. To avoid this, a validation check to stop this has been implemented.

2	After implementing new validation.	<pre> Player 1 name: Invalid input. Player 1 name: </pre>	When a user inserts a blank entry. An exception is displayed to stop empty names from appearing.
3	If a user inserts a valid name	<pre> Player TestUser123, place Destroyer peice. Length 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 SH 1 - Up 2 - Right 3 - Down 4 - Left vc Direction: _ vc Pl vc </pre>	When a user inserts a string. The name is saved and stored onto the player class and shown correctly.

Ship placements

When the users have inserted their usernames. The placements are next. Meaning that multiple inputs will need to be tested. The correct ship needs to be displayed, the direction input needs to be tested as well as the coordinates.

For the validation, only numbers need to be inserted the range as well needs to be specified for the direction. The coordinates need validate the range depending on the grid size.

Test	Description	Evidence	Outcome
1	Test to check what happens if a user tries to enter the wrong coordinates	<pre> X: 0 Number must be within 1 - 10. vc X: asd Input is not a number vc X: Invalid input. </pre>	They are given an error when they try to insert something out of bounds, invalid or blank.

2	Test to see what happens if a user tries to overlap ships.	<pre> Player TestUser123, plac 1 2 3 4 5 6 7 8 9 10 1 - - 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 1 - Up 2 - Right 3 - Down 4 - Left Direction: 3 X: 1 Y: 1 Invalid Placement </pre>	A error is displayed to user the user saying that the placement is invalid.
3	Test to check what happens if the user tries to place the ship in the grid but facing out of bounds	<pre> Player TestUser123, plac 1 2 3 4 5 6 7 8 9 10 1 - - 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 - 9 0 0 0 0 0 0 0 0 0 0 - 10 0 0 0 0 0 0 0 0 0 0 - 1 - Up 2 - Right 3 - Down 4 - Left Direction: 4 X: 1 Y: 2 Invalid Placement </pre>	The same error is shown from test 1 due to the invalid placement.

4	Test to see what happens if a user places ships in valid spaces	<pre> Player Player 1, place Cru 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 2 0 - - 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 - 0 0 0 0 0 6 0 0 0 0 - 0 0 0 0 0 7 0 0 0 0 - 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 1 - Up 2 - Right 3 - Down 4 - Left Direction: Invalid input. </pre>	Ships are shown in the correct position facing the correct direction.
---	---	--	---

Player guesses

When the player guesses, the input needs to validate that the coordinate is correct as well as check if the grid section has already been checked. The testing done will be:

- Letters
- Blank
- Numbers out of range
- Valid input

Test	Description	Evidence	Outcome
1	When a player guesses an already guessed place	<pre> Player Player 2 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 X 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 1 Y: 1 Coordinate already guessed. Try again. </pre>	An error is shown to the user stating that they have already guessed this location and allowed to guess again.

2	If the user tries to guess out of bounds	<pre> X: awd Input is not a number X: -1 Input is not a number X: 0 Number must be within 1 - 10. X: 11 Number must be within 1 - 10. </pre>	When the user tries to insert an invalid input. They are given exceptions depending on the input.
3	What happens when a user hits a ship	<pre> Player Player 1 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 X 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 1 Y: 2 Hit! </pre>	The correct output is shown
4	What happens if a player misses	<pre> Player Player 2 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 X 0 0 0 0 0 0 0 0 0 2 # 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 9 Y: 9 Miss! </pre>	The correct output is shown

5	When all ship tiles are hit. The player wins	<pre>Player 1 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 X X 0 0 0 0 0 0 0 0 2 X X 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 3 Y: 2 Hit! Player 1 WON!!!</pre>	When a player hits all the opponents tiles, the player wins.
---	---	---	--

Grid check

- When the player places a ship and every turn the player guesses, a grid is displayed. The grid is dynamic to the players grid data. Meaning that every section of the grid can be changed depending on ship placements and grid guesses. This means testing needs to be done on guessing and guessing to see if the correct output is being displayed.

Test	Description	Evidence	Outcome
1	Grid shows ships in placements but not in guessing	<pre>Player 1, place Submarine peice. Length 3. 1 2 3 4 5 6 7 8 9 10 1 - - 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 1 - Up 2 - Right 3 - Down 4 - Left Direction: _</pre>	As we can see. When placing ships, the already placed ships are displayed when still placing the other ships. But when the guessing phase is active, the opponent cannot see the placement of the ships. This means that the correct displays are being shown.

		<pre>Player 2 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: _</pre>	
2	Misses are displayed correctly	<pre>Player 2 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 1 Y: 2 Miss! _ Player 2 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 0 2 # 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X:</pre>	As we can tell. The section 1,2 has a '#'. This means that the player missed when guessing.

3	Hits are displayed correctly	<pre>Player 1 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: 1 Y: 1 Hit! -</pre> <pre>Player 1 turn to guess. 1 2 3 4 5 6 7 8 9 10 1 X 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 X: -</pre>	As shown. The player who guessed 1,1 can see that the ship has been hit within the grid.
---	------------------------------	---	--

References

1 - GeeksForGeeks

<https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/>

Goes over the main concepts of OOP. This reference is used for classes / objects

2 – Sumo Logic

Goes over encapsulation within OOP

<https://www.sumologic.com/glossary/encapsulation/>

3 – Stackify

Goes over the concept of Abstraction

<https://stackify.com/oop-concept-abstraction/>

4 – Tutorials Point

Goes over inheritance within c++ and OOP

https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

5 – Medium

Article by Shanika Ediriweera about Polymorphism in OOP

<https://medium.com/@shanikae/polymorphism-explained-simply-7294c8deef7>

Assessment 1 - ZE - 1908686

GRADEMARK REPORT

FINAL GRADE

94/100

GENERAL COMMENTS

Instructor

Classes (25): 24
Polymorphism (10): 8

Game (20): 20

Report (35): 32

Testing (5): 5

Demo (5): 5

Feedback:
=====

Report needed a little more discussion of polymorphism.

Nice example of battleship game.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14



Comment 1

The example here does not demonstrate polymorphism. The banner() method actually hides the method in the parent class rather than overriding it. You have to make it virtual in the parent class.

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26
