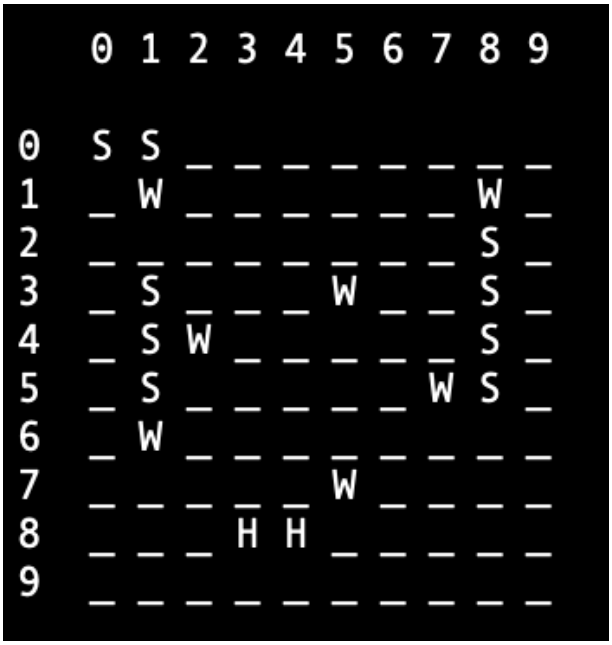


BATTLESHIP

Computer Science Project

Description

A guessing game played by two players whose objective is to sink enemy ships

Brinda Puri & Vibha Masti
12 B

Table of Contents

Objective

The objective of Battleship is to try and sink all of the other player's (enemy's) ships before they sink all of yours. All of the enemy's ships are somewhere on their board. You try and hit them by calling out the coordinates of one of the squares on the board. The enemy also tries to hit your ships by calling out coordinates. Neither you nor the other player can see the other's board so you must try to guess where they are. Each player in the physical game has one grid for placing their ships and another grid to record their guesses.

Placing ships

```
Pthalium's map

   0 1 2 3 4 5 6 7 8 9

0  0 1 1 0 0 0 0 0 0 0
1  0 0 0 0 0 0 0 0 0 0
2  0 0 0 0 0 0 0 1 0 0
3  0 0 0 0 0 1 0 1 0 0
4  0 0 1 0 0 1 0 1 0 0
5  0 0 1 0 0 1 0 1 0 0
6  0 0 1 0 0 1 0 0 0 0
7  0 0 0 0 0 1 0 0 0 0
8  0 0 0 0 0 0 0 0 0 0
9  0 1 1 1 1 1 1 0 0 0
```

Guesses

```
Pthalium: turn 19
Dopeler's current map

   0 1 2 3 4 5 6 7 8 9

0  S S _ _ _ _ _ _ _ _
1  _ W _ _ _ _ _ _ W _
2  _ _ _ _ _ _ _ _ S _
3  _ S _ _ _ W _ _ S _
4  _ S W _ _ _ _ _ S _
5  _ S _ _ _ _ _ W S _
6  _ W _ _ _ _ _ _ _ _
7  _ _ _ _ _ _ W _ _ _
8  _ _ _ H H _ _ _ _ _
9  _ _ _ _ _ _ _ _ _ _
```

Game Rules

The game Battleship is a guessing game played by two people. The game is played on a board of square grids, 10x10.

Each ship occupies a number of consecutive squares on the board, arranged either horizontally or vertically, but not overlapping each other. The lengths of the ships pre-defined in the game as 2,3,4,5 or 6. There are 5 ships in total.

Before the game starts, each player secretly places their ships anywhere on their own playing field **(shipMap)**.

The game begins with the first player entering a pair of guess coordinates. If the guessed coordinates correspond with the location of an enemy ship, the shot is called a hit **(H)**. If, however, the coordinates do not correspond with any enemy ship locations, it is called a miss **(W)**. If a player manages to hit all the grids that a single ship occupies, that ship is said to be sunk **(S)**. If all of a player's ships have been sunk, they lose the game.

Game Design - Data Structures and Data Files

In order to implement the game, numerous classes and structures have been used to organise player and ship data. The following block diagram should explain the relationship between the classes:

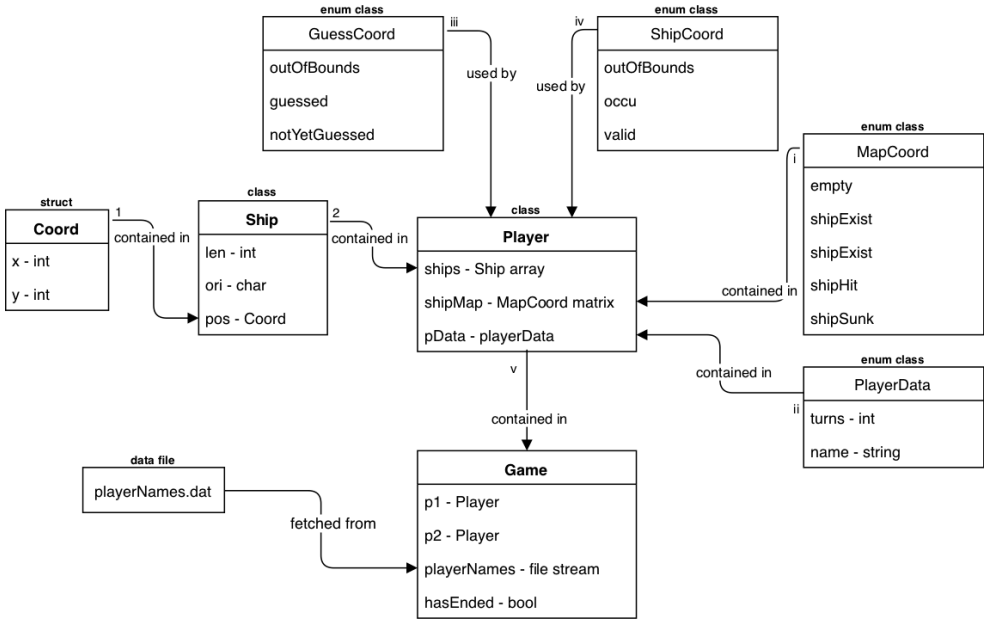


Figure 1

Coord

Type:
Structure

Data:

- Two integers, **x** and **y**, to represent the two coordinates of a point.

Functions:

- **Coord()** – default constructor: generates a **Coord** object and sets **x** and **y** to 0
- **Coord(int, int)** – parameterised constructor: generates a **Coord** object with two integers for **x** and **y**
- **void input()** – inputs **x** and **y**

Ship

Type:
Class

Data:

- **len** – integer that stores ship length
- **ori** – character that stores orientation **('h'/'v')** of a ship
- **pos** – **Coord** that stores starting position of ship

Functions:

- **Ship()**- constructor: generates a **Ship** object and initialises all data members to their default values
- **void setLen(int)** – sets **len** with an integer
- **void input()** – inputs **pos** and **ori**
- **int getLen()** – returns **len**
- **int getPos()** – returns **pos**
- **char getOri()** – returns **ori**

ShipCoord

Type:
Enum class

Data:

- three possible statuses of a coordinate pair for a ship of length **len** – **outOfBounds** (coordinates lie outside **shipMap**), **occu** (coordinates are occupied), and **valid** (coordinates are available)

MapCoord

Type:
Enum class

Data:

- five possible statuses of a coordinate pair with regards to the presence of a ship – **empty** (no ship exists at this point), **shipExist** (a ship exists at this point but it hasn't been guessed yet), **shipHit** (a ship exists at this point and has been guessed/hit), **shipSunk** (a ship exists at this point and has been sunk), **shipMiss** (no ship exists at this point; only water)

GuessCoord

Type:
Enum class

Data:

- three possible statuses of a coordinate pair that has been guessed – **outOfBounds** (the coordinates lie outside the **shipMap**), **guessed** (the coordinate pair has already been guessed), **notYetGuessed** (the coordinate pair has not yet been guessed)

PlayerData

Type:
Structure

Data:

- **name** - a string that stores player name

- **turns** - an integer that stores the number of turns played in a round

player.h macros

Type:
Preprocessor directives

Data:

- **NO_OF_SHIPS** – number of ships that each player has to arrange in their map (set to 5)
- **MAP_SIZE** – size of the player grid, **shipMap**, in number of units (set to 10)
- **STR_LEN** – maximum length of a character array (to prevent the use of magic numbers)
- **INF** – a number that represents infinity, set as **MAP_SIZE**^{2} + 1 (always exceeds the maximum number of turns possible)

Player

Type:
Class

Data:

- **ships[]** – array of **Ship** objects that are to be arranged on the gameboard
- **shipMap[][]** – a matrix of **MapCoord** values, which represents the player's grid
- **pData** – a **PlayerData** object that stores all of a player's data
- **minTurns** – integer that stores the player's best attempt at the game/ least number of turns (default set to **INF**) which is later fetched from a data file

Functions:

- **Player()** – default constructor: sets all **shipMap** values to **MapCoord::empty**, sets the length of each ship as a number from 2 to **NO_OF_SHIPS** + 2
- **void inputName()** – inputs **name** (from **pData**) of a player
- **void inputShips()** – inputs the position and orientation of all **ships** on **shipMap**
- **char symbol(MapCoord)** – returns a symbol corresponding to a **MapCoord** value, which will be printed when viewing the player grid
- **void printFullMap()** – prints a map of symbols of all the coordinates of the **shipMap** that corresponds to a player's grid, showing the locations of all of the player's ships (this function is only called before the opponent begins guessing coordinates)
- **void printGuessMap()** – prints a map of symbols of all the coordinates of the **shipMap** that have been guessed by the opponent, and hides all the remaining coordinates

- **void drawShip(Ship)** – changes **MapCoord** values on the **shipMap** of the coordinates that correspond to the Ship object that is passed on to this function, thereby 'drawing' the map onto the grid
- **bool inMap(Ship)** – returns whether a **Ship** will lie entirely within the map (based on its starting position and orientation)
- **bool mapFree(Ship)** – returns whether a **Ship** lies on coordinates that have not yet been occupied by other Ships, to prevent overlapping
- **ShipCoord getStatus(Ship)** – returns a **ShipCoord** value that describes the validity of a **Ship**'s coordinates (this function was written to prevent a bug that allowed a player to first enter coordinates that lied outside the map and then, when prompted to re-enter, allowed them to enter coordinates that were already occupied by another **Ship**). Every entered coordinate is passed onto this function, which calls both **inMap** and **mapFree** to check the validity of each coordinate and returns a **ShipCoord** value that clearly distinguishes between the two invalid statuses
- **bool isSunk(Ship)** – returns whether a Ship has been sunk by the other player or not by checking its **MapCoord** values on **shipMap**
- **void sink(Ship)** – changes **MapCoord** values of a ship on **shipMap** from hit (**MapCoord::shipHit)** to sunk (**MapCoord::shipSunk)**, thereby 'sinking' the ship
- **bool allShipsSunk()** – returns whether all of a player's ships have been sunk, which is used to check if a game has ended
- **GuessCoord guessValidity(Coord)** – returns a **GuessCoord** value that tells whether a coordinate pair lies outside the **shipMap**, has already been guessed, or has not been guessed at all (created to prevent the same bug that was encountered in checking for **Ship** coordinates)
- **Coord getsHit(Coord)** – changes the **MapCoord** value of the guessed coordinate on the **shipMap** based on whether a ship exists or not. If the coordinate entered is an invalid coordinate, as checked by **guessValidity**, the player is prompted to enter a new coordinate pair which is then returned at the end of the function. This was done inside the function instead of outside to allow for a clearer separation of concerns. We didn't want the **Game** object (outer class instance) to have to check for **Player** object (inner class instances) guess validities
- **char* getName()** – returns player **name** from **pData**
- **int getTurns()** – returns number of **turns** played so far
- **int getMinTurns()** – returns the player's minimum number of turns
- **void increaseTurns()** – increments the number of turns played so far
- **void setMinTurns()** – sets the minimum number of turns (best attempt) of a player from outside, using a data file

playerNames.dat

Type:
Data file

Data:

- stores **PlayerData** structures of every player that has ever played the game, containing only their best attempts (minimum number of turns)

Game

Type:
Class

Data:

- two **Players**, **p1** and **p2**
- **playerNames** – a std::fstream (file stream) that connects to a binary file **playerNames.dat**
- **hasEnded** – a boolean value that stores whether a game has ended (default is false)

Functions:

- **Game()** – default constructor: initialises data members to their default values
- **void inputPlayerData(Player&, Player&)** – inputs name and ships of a player while ensuring that the two players do not have the same name
- **void inputPlayers()** – inputs player data for the two players while accounting for leftover whitespaces in the cin stream
- **void printRules()** – prints the game rules of battleship
- **void checkFile(Player&)** – checks the file for a player's name and best attempt
- **void playRound(Player&, Player&)** – executes one round of playing the game, which involves a player that makes the shot (**madeHit**) and the player that gets shot at (**gotHit**)
- **bool inPlay()** – returns whether the game is still in play or not (returns negative of **hasEnded**)
- **void saveBestTurns(char*, int)** – saves a new best attempt for a player onto the data file **playerNames.dat**

Verifying Player Inputs

Inputting Player Names

In this two-player game, each player's name is treated as their username and their personal bests are stored in a data file **playerNames.dat**. Once the programme is run, the first player is asked to enter their name, and the file is scanned for the entered name. If the name exists, the player's personal best is displayed on the screen, and if the name doesn't exist, it is added to the file with a personal best number of turns set to **INF**. Once the second player enters their name, it is compared to the first player's name. If the two player names entered are exactly the same (case-sensitive), then the second player is prompted to re-enter their name. Only after the name has been verified will the programme begin scanning the data file once again.

Inputting Ship Coordinates

To input ship co-ordinates, the players are each asked to enter the starting position and orientation of their ships. The coordinates and orientation are sent to a checking function **getStatus** and a **ShipCoord** (refer *Figure 1*) value is returned based on the following:

outOfBounds (ship lies outside the map) → 0
occu (occupied by another ship) → 1
valid (available) for ship → 2

Only if **ShipCoord::valid** is returned, the ship will be placed at the coordinates entered. Until then, the user is prompted to re-enter the coordinates and orientation.

Initialising Player Map

The **shipMap** acts as the gameboard of the players. Each coordinate pair value in the **shipMap** is given a **MapCoord** value based on the following conditions (everything is set to **MapCoord::empty** by default):

empty (no ship occupies the coordinates) → 0
shipExist (a ship exists but has not yet been guessed) → 1
shipHit (a ship exists and has been hit)→ 2
shipSunk (a ship exists and has been sunk) → 3
shipMiss (no ship exists; water was hit) → 4

When a player inputs a ship, the coordinates occupied by the ship attain a value of **MapCoord::shipExist** on the **shipMap**. These **MapCoord** values are the key to creating this version of the game as they convey all of the information of a point.

Inputting Guess Coordinates

Each round of this game involves players guessing a coordinate pair on their opponent's gameboard. Therefore, it is important that we ensure that there are no repeated guesses. Each coordinate pair is passed onto a checking function **guessValidity** that returns one of the following **GuessCoord** values based on the coordinate pair's **MapCoord** value:

outOfBounds (coordinates outside the map) → 0
guessed (already been guessed once) → 1
notYetGuessed → 2

The function **guessValidity** maps the following **MapCoord** values to **GuessCoord** values:

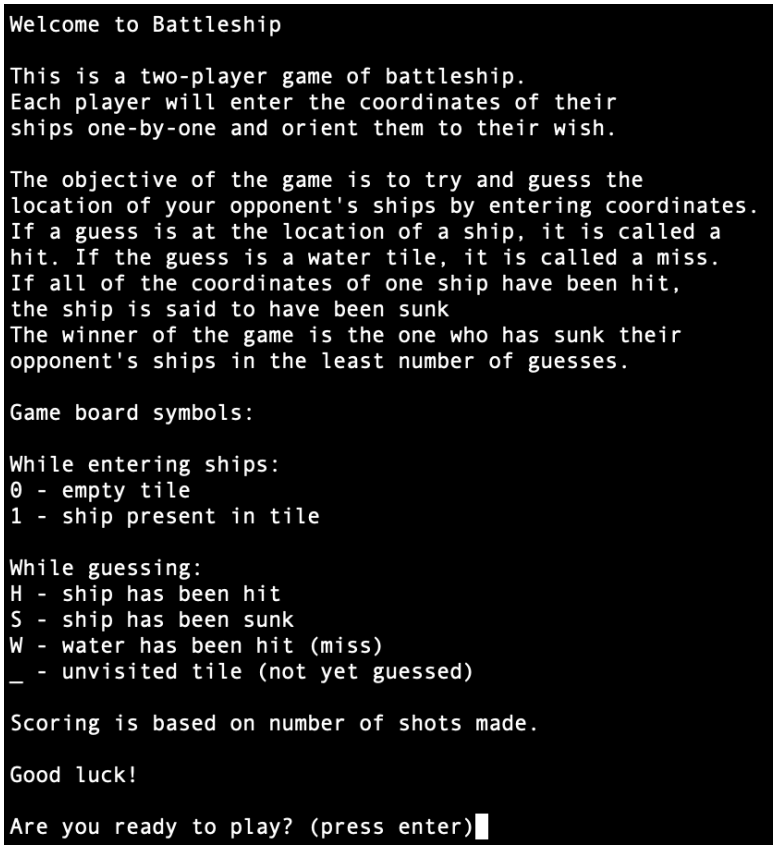
empty - notYetGuessed
shipExist - notYetGuessed
shipHit - guessed
shipSunk - guessed
shipMiss – guessed

Only if the guessed coordinate pair returns **GuessCoord::notYetGuessed**, the game progresses. If the coordinates lie outside the **shipMap**, **GuessCoord::outOfBounds** is returned, and if the coordinates have already been guessed by the player before, **GuessCoord::guessed** is returned and the player is asked to re-enter their coordinates until **guessValidity** returns **GuessCoord::notYetGuessed**.

Gameplay

Welcome Screen

The programme starts with a welcome screen that prints the rules of the game and waits for the two players to get ready. Once a player presses enter, the game begins.



Inputting Player Data

Starting with Player one, the players are asked to first enter their name and then to enter ship coordinates.

```
Player one

   0 1 2 3 4 5 6 7 8 9

0  0 0 0 0 0 0 0 0 0 0
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

Enter player name: Pthalium

Hello, Pthalium! First time playing?
```

```
Details of ship 1 (length: 2 units)
Starting position (coordinates):
X: 1
Y: 0
Orientation (v/h): h


Updated map:


   0 1 2 3 4 5 6 7 8 9

0  0 1 1 0 0 0 0 0 0 0
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
```

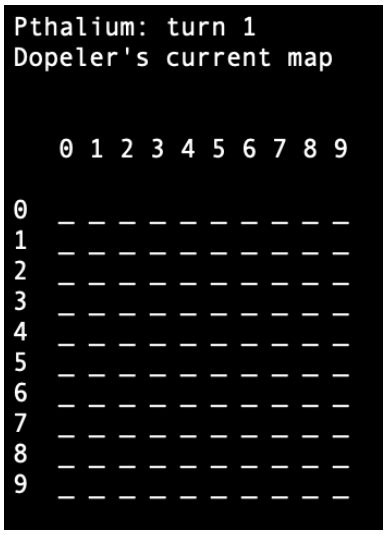
After all of player one's ships have been entered, their entire map is shown and the programme waits for player one to call player two, after which the same thing is repeated. Players one and two are not allowed to view their opponents' maps, which is why they are asked to press enter before calling the next player.

```
Pthalium's map


   0 1 2 3 4 5 6 7 8 9

0  0 1 1 0 0 0 0 0 0 0
1  0 0 0 0 0 0 0 0 0 0
2  0 0 0 0 0 0 0 0 1 0 0
3  0 0 0 0 0 0 1 0 1 0 0
4  0 0 1 0 0 1 0 1 0 0
5  0 0 1 0 0 1 0 1 0 0
6  0 0 1 0 0 1 0 0 0 0
7  0 0 0 0 0 0 1 0 0 0 0
8  0 0 0 0 0 0 0 0 0 0 0
9  0 1 1 1 1 1 1 1 0 0 0

Call player 2: (press enter)
```

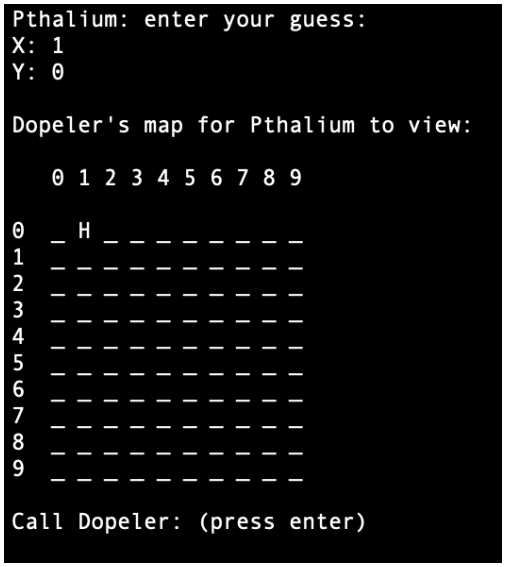
```
Dopeler's map


   0 1 2 3 4 5 6 7 8 9

0  1 1 0 0 0 0 0 0 0 0
1  0 0 0 0 0 0 0 0 0 0
2  0 0 0 0 0 0 0 0 1 0
3  0 1 0 0 0 0 0 0 1 0
4  0 1 0 0 0 0 0 0 1 0
5  0 1 0 0 0 0 0 0 1 0
6  0 0 0 1 1 1 1 1 0 0
7  0 0 0 0 0 0 0 0 0 0
8  0 0 0 1 1 1 1 1 1 0
9  0 0 0 0 0 0 0 0 0 0

Call Pthalium: (press enter)
```

Guessing Coordinates

After both players have successfully entered their ships, player one is called to start playing the game. An empty board of player two is displayed and player one is asked to make their first guess.



If the guess is a hit, the letter ‘H’ is displayed on the board, and if the coordinate is a miss, the letter ‘W’ (for water) is displayed on the board. The programme then waits for player two to come to the board and press enter, before they can make their first guess.



If all the coordinates of a ship have been hit, the ship is then sunk and the letter ‘S’ is displayed on the board.

```
Dopeler: enter your guess:
X: 7
Y: 5

Pthalium's map for Dopeler to view:

   0 1 2 3 4 5 6 7 8 9

0  _ _ _ _ _ _ _ _ _ _
1  _ _ _ _ _ _ _ W _ _
2  _ _ _ _ _ _ _ S _ _
3  _ _ _ _ _ _ _ S _ _
4  _ _ _ _ _ _ _ S _ _
5  _ _ _ _ _ _ _ S _ _
6  _ _ _ _ _ _ _ _ _ _
7  _ _ _ _ _ _ _ _ _ _
8  _ _ _ _ _ _ _ _ _ _
9  _ _ _ _ _ _ _ _ _ _

Call Pthalium: (press enter)
```

If a player sinks all of their opponent’s ships, that player wins the game.

```
Pthalium: enter your guess:
X: 3
Y: 6

Dopeler's map for Pthalium to view:

   0 1 2 3 4 5 6 7 8 9

0  S S _ _ _ _ _ _ _ _
1  _ W _ _ W _ _ _ W _
2  _ _ _ _ _ _ _ _ S _
3  _ S _ _ _ W _ _ S _
4  _ S W _ _ _ _ _ S _
5  _ S _ _ _ _ _ W S _
6  W W _ S S S S S W _
7  _ _ _ _ _ W _ _ _ _
8  _ _ W S S S S S S _
9  _ _ _ _ _ _ _ _ _ _

Congratulations, Pthalium!
You have won the game in 31 turns
New best attempt!
Hard luck, Dopeler!
```