BATTLE SHIPS

Project submitted to the

SRM University – AP, Andhra Pradesh

Submitted in partial fulfilment of the requirement for the award of the degree of

Bachelor of Technology in

Computer Science and Engineering

School of Engineering and Sciences

Submitted By

M.SOHITH(AP23110010729)

V.SURYA ANISH(AP23110010783)

N.PANVEE(AP23110010749)

J.KARTHIK(AP23110010740)

Under the guidance of

CH.MARY

Department of Computer Science and Engineering

SRM University,AP

Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240 [Month, Year]

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **"Project title"** is being submitted by

M.SOHITH(AP23110010729)

V.SURYA ANISH(AP23110010783)

N.PANVEE(AP23110010749)

J.KARTHIK(AP23110010740)

students of Department of Computer Science and Engineering, SRM University,AP, in partial fulfilment of the requirement for the degree of **"B.Tech(CSE)"** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor

Signature of Head of the Dept.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **CH MARY**, Department of Computer Science & Engineering, SRM University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

M.SOHITH(AP23110010729)

V.SURYA ANISH(AP23110010783)

N.PANVEE(AP23110010749)

J.KARTHIK(AP23110010740)

TABLE OF CONTENTS

INTRODUCTION

The game **Battleships** has entertained players for decades, evolving from pen-and-paper to digital platforms. The game's popularity stems from its blend of strategy, logic, and chance, making it an ideal candidate for a programming project. This project focuses on implementing Battleships in C++ as a console-based application.

The primary motivation behind this project is to apply core programming concepts in a practical and engaging way. By leveraging object-oriented programming (OOP), dynamic memory management, and structured programming, this project provides valuable insights into game design and development.

The scope of this project includes creating a functional two-player version of Battleships with features like dynamic board setup, real-time feedback on moves, and endgame conditions. The purpose is to demonstrate programming skills and create a foundation for future enhancements.

LITERATURE REVIEW

Previous implementations of Battleships often utilize Python, Java, or C#, leveraging graphical user interfaces. However, console-based versions in C++ provide a deeper understanding of game mechanics and algorithms. Various resources, such as textbooks on C++ game programming and online tutorials, informed the development process. Notably, prior research on game theory and logic influenced the strategy-building features of the application. Open-source implementations of Battleships in other languages inspired ideas for structuring the game board and handling user input. A study of game theory helped enhance strategic gameplay elements. The literature highlighted the gap in C++ implementations of Battleships that focus on modularity, scalability, and user experience. This project aims to address this gap.

PROBLEM STATEMENT

Developing a console-based version of Battleships involves addressing several challenges. First, the game requires the creation of a dynamic grid system that supports flexible ship placement and accurate attack tracking. Ensuring robust error handling is critical to prevent invalid inputs, such as attacking the same position repeatedly or placing ships outside the grid boundaries. Lastly, the game should deliver an engaging experience for players, providing real-time feedback on their moves and ensuring that the rules are upheld throughout the gameplay. This project aims to solve these problems while adhering to the traditional rules of Battleships and maintaining a high standard of software quality.

OBJECTIVES

The objective of this project is to create a fully functional console-based implementation of Battleships that adheres to the original gameplay rules. This includes designing a dynamic and interactive interface that allows players to easily place their ships and make strategic moves. Another key objective is to leverage OOP principles to ensure the modularity and scalability of the code. Error handling mechanisms will be integrated to prevent invalid actions and enhance the overall user experience. Finally, the project aims to serve as a foundation for potential future expansions, such as the inclusion of artificial intelligence or graphical elements.

METHDOLOGY

This project was developed using C++ due to its robust features for handling OOP and memory management. Visual Studio Code and Code::Blocks were used as integrated development environments, and Git was employed for version control.
The development process began with planning the game rules and features. Detailed flowcharts and pseudocode were created to map the gameplay logic. A class-based approach was chosen for implementation, with entities like Player, Ship, and Board designed as separate components. The Player class manages individual player actions, while the Board class handles grid operations, and the Ship class encapsulates the properties of each ship. Rigorous testing was conducted to ensure the application handled all edge cases, such as invalid inputs and overlapping ships.

IMPLEMENTATION

The implementation of the game follows the classic rules of Battleships. Each player has a grid where they place their fleet of ships, and they take turns guessing the location of the opponent's ships. The game begins with a setup phase, where players arrange their ships on the grid. The grid is dynamically generated, with ship placement validated to prevent overlaps and out-of-bound errors.
Gameplay proceeds with players taking turns to attack specific grid coordinates. Feedback is provided immediately, indicating whether the attack was a hit or a miss. If a ship is fully destroyed, the game notifies the players. The game ends when all ships of one player have been sunk, and the winner is declared.
The code structure emphasizes modularity, with clear separation of functionality between classes. For example, the Ship class tracks the health and status of each ship, while the Board class maintains the positions of ships and records attack outcomes.

RESULT AND ANALYSIS

The project successfully meets its objectives by creating a functional Battleships game. All core features, including ship placement, turn-based gameplay, and endgame conditions, work as intended. The game provides an interactive experience through clear prompts and feedback. Players are prevented from making invalid moves, ensuring a smooth flow of gameplay.

Testing revealed that the game handled edge cases effectively, such as overlapping ship placement and repeated attacks on the same position. Performance metrics indicate that the game operates efficiently, with quick response times for all actions. Players found the interface intuitive, even in its console-based form, highlighting the success of the design.

DISCUSSION

This project demonstrates the potential of C++ for developing interactive applications. The use of OOP principles allowed for a modular and scalable design, making it easier to extend the game in the future. However, the console-based interface, while functional, limits the game's appeal compared to versions with graphical interfaces. Future enhancements could address this limitation by integrating a GUI using libraries such as SFML or Qt. Additionally, incorporating an AI opponent would allow for single-player gameplay, further expanding the project's scope and audience.

CONCLUSION

In conclusion, this project successfully implements a console-based version of Battleships using C++. The game fulfills its objectives, providing an engaging and error-free experience for players. It serves as a practical demonstration of core programming concepts and lays the groundwork for further development. By addressing the challenges of dynamic grid management, robust error handling, and intuitive gameplay, this project contributes to the understanding of game development in C++.

FUTURE SCOPE

This project has significant potential for expansion. Adding a graphical user interface would make the game more appealing and accessible to a wider audience. Incorporating artificial intelligence would enable single-player gameplay, allowing users to compete against a computer-controlled opponent. Multiplayer functionality over a network could further enhance the game's reach. Additionally, custom game modes and adjustable grid sizes could provide a more personalized experience for players.

REFERANCES

For timer we gone through

geeks for geeks https ://www.geeksforgeeks.org/how-to-add-timed-delay-in-cpp/

- Emplace back https://www.geeksforgeeks.org/vectoremplace_back-c-stl/
- Exception Handling https://www.geeksforgeeks.org/exception-handling-c/
- Template https://www.geeksforgeeks.org/templates-cpp/
- Scope resolution https://www.geeksforgeeks.org/scope-resolution-operator-in-c/

CODE

```
#include <iostream>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For time()
using namespace std;

const int gridSize = 10;
const char emptyCell = '-';
const char shipCell = 'S';
const char hitCell = 'H';
const char missCell = 'M';

void initializeGrid(char grid[gridSize][gridSize]) {
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridSize; j++) {
            grid[i][j] = emptyCell;
        }
    }
}

void printGrid(char grid[gridSize][gridSize], bool hideShips = false) {
    cout << "  A B C D E F G H I J" << endl;
    for (int i = 0; i < gridSize; i++) {
        cout << i + 1 << " ";
        for (int j = 0; j < gridSize; j++) {
            if (hideShips && grid[i][j] == shipCell)
                cout << emptyCell << " ";
            else
                cout << grid[i][j] << " ";
        }
        cout << endl;
```

```
    }
}

bool isValidPlacement(char grid[gridSize][gridSize], int row, int col, int length, char
direction) {
    if (direction == 'H') {
        if (col + length > gridSize) return false;
        for (int i = 0; i < length; i++) {
            if (grid[row][col + i] != emptyCell) return false;
        }
    } else if (direction == 'V') {
        if (row + length > gridSize) return false;
        for (int i = 0; i < length; i++) {
            if (grid[row + i][col] != emptyCell) return false;
        }
    }
    return true;
}

void placeShip(char grid[gridSize][gridSize], int length) {
    int row, col;
    char direction;
    do {
        row = rand() % gridSize;
        col = rand() % gridSize;
        direction = rand() % 2 == 0 ? 'H' : 'V'; // Horizontal or Vertical
    } while (!isValidPlacement(grid, row, col, length, direction));

    if (direction == 'H') {
        for (int i = 0; i < length; i++) {
            grid[row][col + i] = shipCell;
```

```
        }
    } else {
        for (int i = 0; i < length; i++) {
            grid[row + i][col] = shipCell;
        }
    }
}

void placeAllShips(char grid[gridSize][gridSize]) {
    placeShip(grid, 5); // Carrier
    placeShip(grid, 4); // Battleship
    placeShip(grid, 3); // Cruiser
    placeShip(grid, 3); // Submarine
    placeShip(grid, 2); // Destroyer
}

bool takeShot(char grid[gridSize][gridSize], int row, int col) {
    if (grid[row][col] == shipCell) {
        grid[row][col] = hitCell;
        cout << "It's a hit!" << endl;
        return true;
    } else if (grid[row][col] == emptyCell) {
        grid[row][col] = missCell;
        cout << "It's a miss!" << endl;
        return false;
    } else {
        cout << "You already fired there!" << endl;
        return false;
    }
}
```

```
bool allShipsSunk(char grid[gridSize][gridSize]) {
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridSize; j++) {
            if (grid[i][j] == shipCell) return false;
        }
    }
    return true;
}

void playerTurn(char enemyGrid[gridSize][gridSize]) {
    string shot;
    int row, col;
    cout << "Enter your shot (e.g., A5): ";
    cin >> shot;

    col = shot[0] - 'A'; // Convert column letter to index
    row = stoi(shot.substr(1)) - 1; // Convert row number to index

    if (row < 0 || row >= gridSize || col < 0 || col >= gridSize) {
        cout << "Invalid coordinates! Try again." << endl;
        playerTurn(enemyGrid); // Recursive call if input is invalid
    } else {
        takeShot(enemyGrid, row, col);
    }
}

void computerTurn(char playerGrid[gridSize][gridSize]) {
    int row, col;
    do {
        row = rand() % gridSize;
        col = rand() % gridSize;
```

```
    } while (playerGrid[row][col] == hitCell || playerGrid[row][col] == missCell);

    cout << "Computer fires at: " << char('A' + col) << row + 1 << endl;
    takeShot(playerGrid, row, col);
}

int main() {
    srand(time(0));  // Seed random number generator

    char playerGrid[gridSize][gridSize], computerGrid[gridSize][gridSize];
    initializeGrid(playerGrid);
    initializeGrid(computerGrid);

    // Place computer's ships
    placeAllShips(computerGrid);

    // For demo purposes, you can print the computer's grid to see ship locations
(optional)
    // cout << "Computer's grid (hidden):" << endl;
    // printGrid(computerGrid);

    // Place player's ships manually (could be randomized or manually entered)
    cout << "Placing player's ships..." << endl;
    placeAllShips(playerGrid);

    // Game loop
    while (true) {
        cout << "\nPlayer's Grid:" << endl;
        printGrid(playerGrid);
        cout << "\nComputer's Grid (your shots):" << endl;
        printGrid(computerGrid, true);
```

```
        // Player's turn
        playerTurn(computerGrid);
        if (allShipsSunk(computerGrid)) {
            cout << "You sank all the computer's ships! You win!" << endl;
            break;
        }

        // Computer's turn
        computerTurn(playerGrid);
        if (allShipsSunk(playerGrid)) {
            cout << "The computer sank all your ships! You lose!" << endl;
            break;
        }
    }

    return 0;
}
```

SAMPLE OUTPUTS