PA3, Battleship Design Document
Design Document
Authors:
Trevor Kaufman
Aidan Madonna

Document Revision History

Date	Version	Description of changes	Author
26/4/2024	0.0	Initial draft	Trevor Kaufman
28/4/2024	0.1	Completion of draft/design doc	Trevor Kaufman
28/4/2024	0.2	UML Diagrams	Aidan Madonna

Battleship Game Technical Document

Introduction

The programming problem involves designing and implementing a Battleship game in C++. The intended audience for this documentation is software developers who are tasked with building the Battleship game program. The goal is to create a functional and efficient game that allows a single user to play against an AI opponent.

1.1 Design Objectives

The program should hide the complexity of ship placement and board management from the end user while providing a user-friendly interface for gameplay. The end user should be able to access functionalities such as starting the game, placing ships, and making guesses during gameplay. The overall scope of the design includes implementing turn-based gameplay, ship placement, guessing coordinates, and determining the outcome of each turn.

2. Design Overview

2.1 Constraints and Assumptions

- The program must adhere to the given constraints:

        - Limited use of libraries: stdlib, time, and iostream.
        - No use of vectors.
        - At least 4 classes, including an abstract base class.
        - At least 1 operator overload and insertion stream overload for every class.
        - At least 1 template function for generalization.
        - Dynamic allocation of objects and arrays.
        - Pass by reference for deep copies.
    - Design choices:
        - Use of no graphical interface to display plain text.
        - Template function for generalization.
        - Deep copy for dynamic allocation.

3. Structural Design

3.1 Design Explanation and Rationale

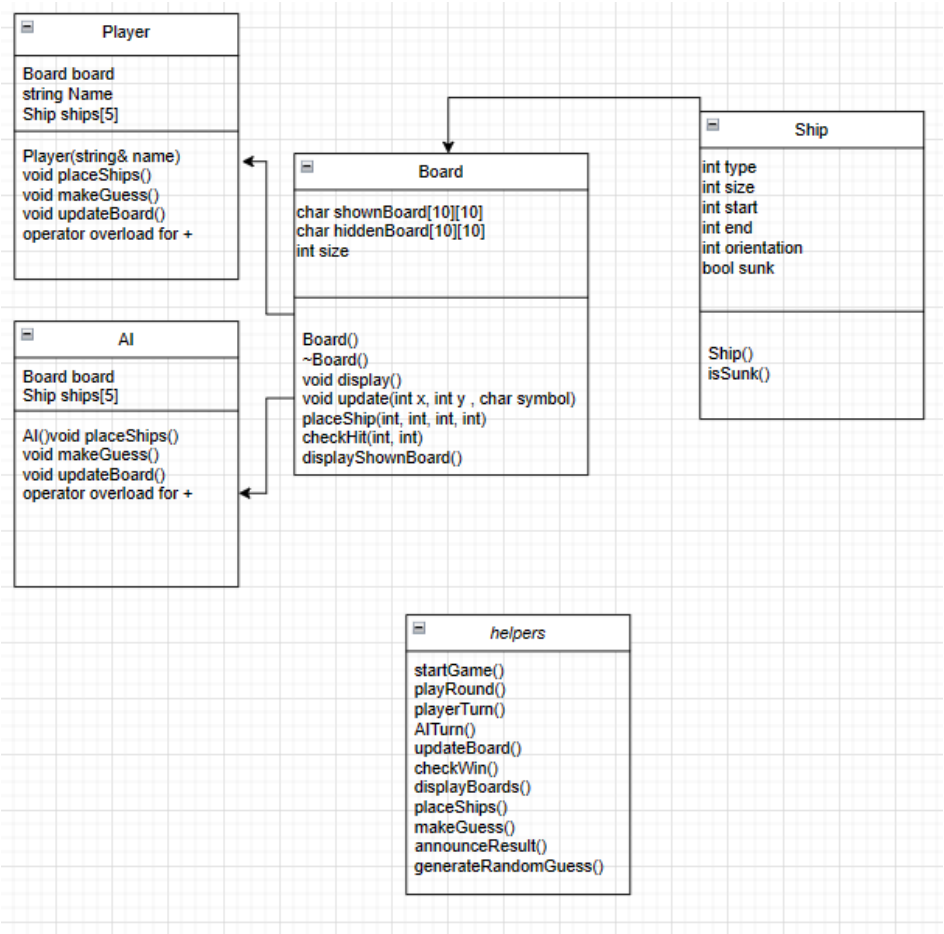
The program is designed using multiple classes to encapsulate different functionalities:

- **Board**: Manages the game board.
- **Ship**: Represents a ship with its properties.
- **Player**: Manages the player's ships and interactions with the board.
- **AI**: Implements AI functionality for ship placement and guessing.

The classes interact to facilitate gameplay, including ship placement, guessing coordinates, and determining hits and misses.

3.2 Class Diagram

Figure 1: Class diagram showing interactions between classes in the Battleship game.



3.3 Class Descriptions

3.3.1 Class: Board

Purpose: Manages the game board for players.

Constraints: None.

3.3.1.1 Attribute Descriptions

- **shownBoard**: Pointer to a 2D array representing the game board user will see.
- **hiddenBoard**: Pointer to a 2D array representing the program works with.
- **size**: Size of the board.

3.3.1.2 Method Descriptions

- **Board()**: Constructor to initialize the board.
- **~Board()**: Destructor to deallocate memory.
- **void display()**: Displays the current state of the board.

- **void update(int x, int y, char symbol)**: Updates the board with the specified symbol at the given coordinates.

3.3.2 Class: Ship

Purpose: Represents a ship with its properties.

Constraints: None.

3.3.2.1 Attribute Descriptions

- **type**: Type of the ship.
- **size**: Size of the ship.
- **start**: Start coordinates of the ship.
- **end**: End coordinates of the ship.
- **orientation**: orientation of the ship.
- **sunk**: Signify if the ship is sunk or still alive.

3.3.2.2 Method Descriptions

- **Ship()**: Constructor to initialize the ship.
- **bool isSunk(const Board& opponentBoard)**: Checks if the ship is sunk based on its coordinates on the opponent's board.

3.3.3 Class: Player

Purpose: Manages the player's ships and interactions with the board.

Constraints: None.

3.3.3.1 Attribute Descriptions

- **name**: Name of the player.
- **board**: Player's game board.
- **ships**: Array of ships owned by the player.

3.3.3.2 Method Descriptions

- **Player(const std::string& name)**: Constructor to initialize the player.
- **void placeShips()**: Allows the player to place their ships on the board.
- **Coordinate makeGuess()**: Prompts the player to make a guess.
- **void updateBoard()**: Updates the player's board based on the guess result.

3.3.4 Class: AI

Purpose: Implements AI functionality for ship placement and guessing.

Constraints: None.

3.3.4.1 Attribute Descriptions

- **board**: AI's game board.
- **ships**: Array of ships owned by the AI.

3.3.4.2 Method Descriptions

- **AI()**: Constructor to initialize the AI.
- **void placeShips()**: Randomly places ships on the board.
- **Coordinate makeGuess()**: Generates a random guess.
- **void updateBoard(const Coordinate& guess, char symbol)**: Updates the AI's board based on the guess result.

3.4 Main and Helpers Descriptions

1. void startGame()

- **Return Type:** void
- **Parameters:** None
- **Return Value:** None
- **Pre-condition State:** None
- **Post-condition State:** Game is initialized and gameplay begins.
- **Class Objects Read/Used:** Player, AI
- **Method Description:** Initiates the game and manages the flow of gameplay, including starting rounds and determining the winner.
- **Call Methods:** playRound()

2. void playRound(Player& player, AI& ai)

- **Return Type:** void
- **Parameters:** player - Player object, ai - AI object
- **Return Value:** None
- **Pre-condition State:** Game has started, player and AI objects are initialized.
- **Post-condition State:** One round of gameplay is completed.
- **Class Objects Read/Used:** Player, AI
- **Method Description:** Manages a single round of the game, including turns for each player and updating the game state.
- **Call Methods:** playerTurn(), AITurn()

3. void playerTurn(Player& player, AI& ai)

- **Return Type:** void
- **Parameters:** player - Player object, ai - AI object
- **Return Value:** None
- **Pre-condition State:** Player's turn has started, player and AI objects are initialized.
- **Post-condition State:** Player's turn is completed, and the board is updated.
- **Class Objects Read/Used:** Player, AI
- **Method Description:** Handles the player's turn, including prompting for guesses and updating the board based on the guess result.
- **Call Methods:** makeGuess(), updateBoard()

4. void AITurn(AI& ai, Player& player)

- **Return Type:** void
- **Parameters:** ai - AI object, player - Player object
- **Return Value:** None
- **Pre-condition State:** AI's turn has started, player and AI objects are initialized.
- **Post-condition State:** AI's turn is completed, and the board is updated.
- **Class Objects Read/Used:** AI, Player
- **Method Description:** Manages the AI's turn, including generating a random guess and updating the board based on the guess result.
- **Call Methods:** makeGuess(), updateBoard()

5. void updateBoard(Board& board, const Coordinate& guess, char symbol)

- **Return Type:** void
- **Parameters:** board - Board object, guess - Coordinate object, symbol - char
- **Return Value:** None
- **Pre-condition State:** Board object is initialized, guess is valid.
- **Post-condition State:** Board is updated with the guess result.
- **Class Objects Read/Used:** Board
- **Method Description:** Updates the board based on the guess result (hit or miss).
- **Call Methods:** Board::update()

6. bool checkWin(Player& player, AI& ai)

- **Return Type:** bool
- **Parameters:** player - Player object, ai - AI object
- **Return Value:** True if the game is over, false otherwise.
- **Pre-condition State:** Player and AI objects are initialized.

- **Post-condition State:** None
- **Class Objects Read/Used:** Player, AI
- **Method Description:** Checks if the game is over by determining if all of the opponent's ships are sunk.
- **Call Methods:** Ship::isSunk()

7. void displayBoards(const Player& player, const AI& ai)

- **Return Type:** void
- **Parameters:** player - Player object, ai - AI object
- **Return Value:** None
- **Pre-condition State:** Player and AI objects are initialized.
- **Post-condition State:** None
- **Class Objects Read/Used:** Player, AI
- **Method Description:** Displays the updated boards at the start of each user turn.
- **Call Methods:** Board::display()

8. void placeShips(Player& player)

- **Return Type:** void
- **Parameters:** player - Player object
- **Return Value:** None
- **Pre-condition State:** Player object is initialized.
- **Post-condition State:** Player's ships are placed on the board.
- **Class Objects Read/Used:** Player
- **Method Description:** Allows the player to place their ships on the board.
- **Call Methods:** Player::placeShips()

9. Coordinate makeGuess()

- **Return Type:** Coordinate
- **Parameters:** None
- **Return Value:** Coordinate object representing the guess.
- **Pre-condition State:** None
- **Post-condition State:** None
- **Class Objects Read/Used:** None
- **Method Description:** Prompts the player to make a guess and returns the coordinates.
- **Call Methods:** None

10. void announceResult(bool isHit)

- **Return Type:** void
- **Parameters:** isHit - bool
- **Return Value:** None
- **Pre-condition State:** None
- **Post-condition State:** None
- **Class Objects Read/Used:** None
- **Method Description:** Announces the result of the guess (hit or miss) to the player.
- **Call Methods:** None

11. Coordinate generateRandomGuess()

- **Return Type:** Coordinate
- **Parameters:** None
- **Return Value:** Coordinate object representing the random guess.
- **Pre-condition State:** None
- **Post-condition State:** None
- **Class Objects Read/Used:** None
- **Method Description:** Generates a random guess for the AI's turn and returns the coordinates.
- **Call Methods:** None
-