BATTLESHIP

Design Document

Cody Kelly

101886601

November 26th, 2017

Problem Description

This is a one or two player game of Battleship. Played on four 10 by 10 grids, ships of various lengths and orientations are placed on two “ship” grids at the start. The objective of the player is to sink all of the AI’s ships before the AI sinks the player’s by utilizing the other two “planning” grids to coordinate attacks.

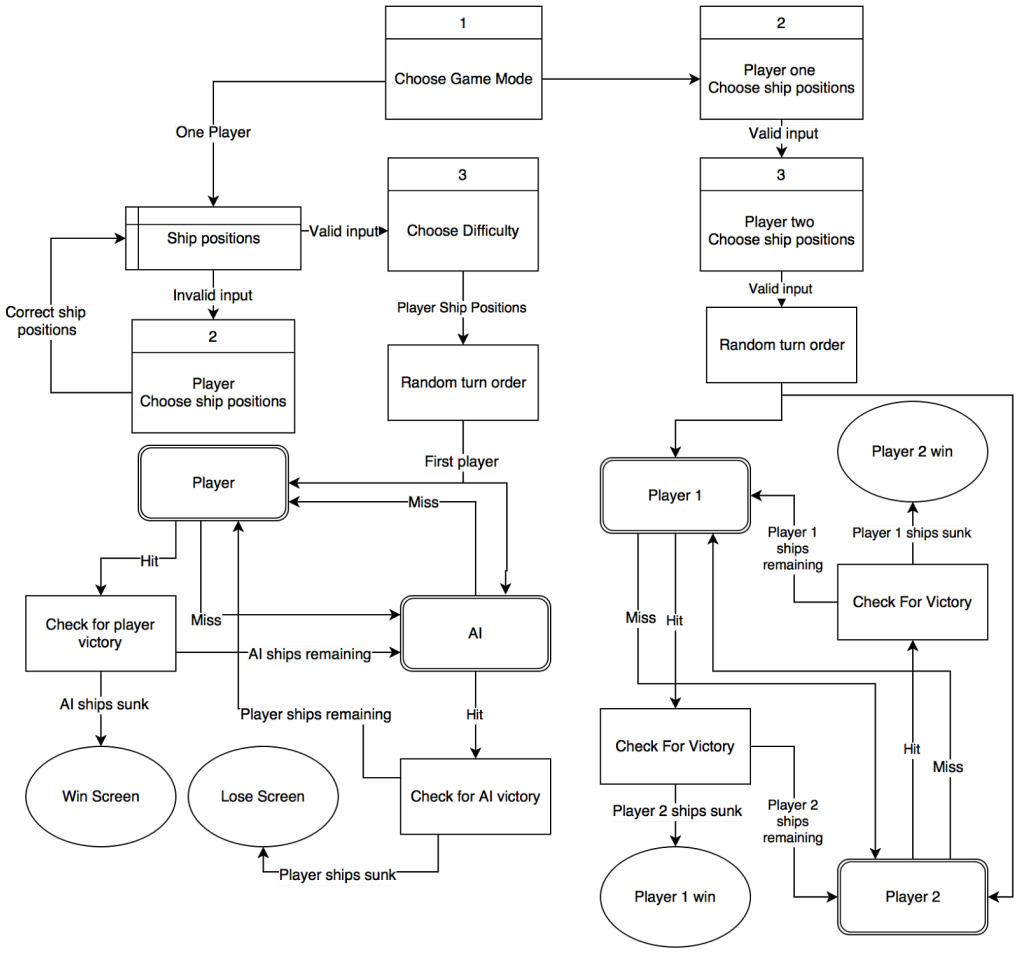
There are two modes this game can be played in:

Classic mode:
Each player gives a coordinate of attack on their turn, with the goal of sinking all of their opponent’s ships. The first player to sink the other’s ships wins.

Salvo mode:
In this mode, each player shoots as many shots as they have boats. If one of their fleet is sunk, they have one less shot in their salvo. This is recommended for more advanced players.

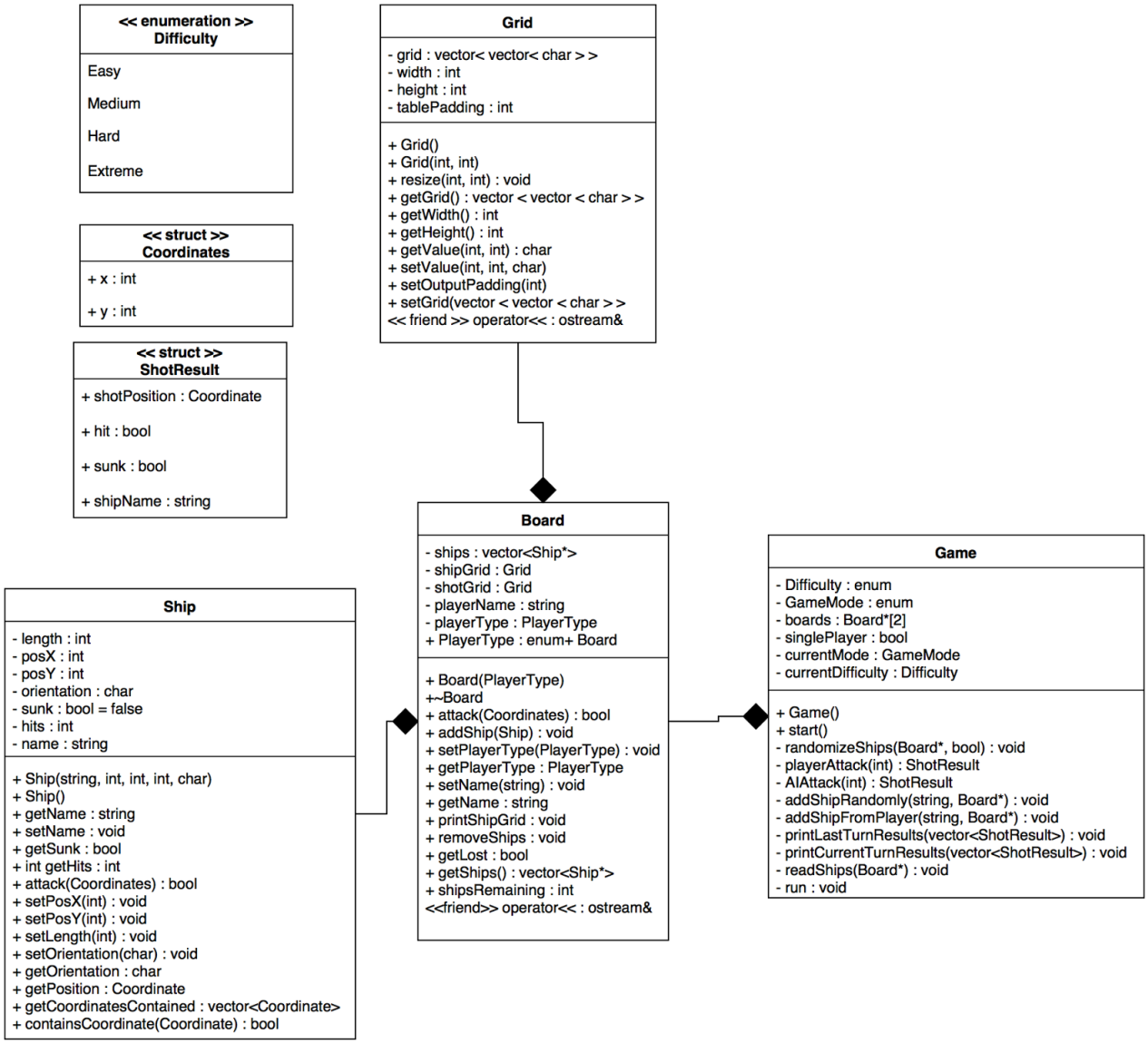
Overall Design Architecture

A main menu will ask the user what game mode they would like to play: one player or two player. If one player, a ship position input function will read in ship positions from a file, clarifying with prompts if any positions are invalid. Then the program will loop, asking the AI and player for attack coordinates, showing an appropriate outcome screen if either wins.



If, however, the player selects a two-player game mode, the program will randomize ship positions for the two players until they are satisfied. Then the program will loop, asking each player for attack coordinates until either players' boats are completely sunk, whereupon the program will display an appropriate outcome screen.

Class Diagram


<< enumeration >>
Difficulty
Easy
Medium
Hard
Extreme
<< struct >>
Coordinates
+ x : int
+ y : int
<< struct >>
ShotResult
+ shotPosition : Coordinate
+ hit : bool
+ sunk : bool
+ shipName : string
Grid
- grid : vector< vector< char > >
- width : int
- height : int
- tablePadding : int
+ Grid()
+ Grid(int, int)
+ resize(int, int) : void
+ getGrid() : vector < vector < char > >
+ getWidth() : int
+ getHeight() : int
+ getValue(int, int) : char
+ setValue(int, int, char)
+ setOutputPadding(int)
+ setGrid(vector < vector < char > >
<< friend >> operator<< : ostream&
Board
- ships : vector<Ship*>
- shipGrid : Grid
- shotGrid : Grid
- playerName : string
- playerType : PlayerType
+ PlayerType : enum+ Board
+ Board(PlayerType)
+~Board
+ attack(Coordinates) : bool
+ addShip(Ship) : void
+ setPlayerType(PlayerType) : void
+ getPlayerType : PlayerType
+ setName(string) : void
+ getName : string
+ printShipGrid : void
+ removeShips : void
+ getLost : bool
+ getShips() : vector<Ship*>
+ shipsRemaining : int
<<friend>> operator<< : ostream&
Ship
- length : int
- posX : int
- posY : int
- orientation : char
- sunk : bool = false
- hits : int
- name : string
+ Ship(string, int, int, int, char)
+ Ship()
+ getName : string
+ setName : void
+ getSunk : bool
+ int getHits : int
+ attack(Coordinates) : bool
+ setPosX(int) : void
+ setPosY(int) : void
+ setLength(int) : void
+ setOrientation(char) : void
+ getOrientation : char
+ getPosition : Coordinate
+ getCoordinatesContained : vector<Coordinate>
+ containsCoordinate(Coordinate) : bool
Game
- Difficulty : enum
- GameMode : enum
- boards : Board*[2]
- singlePlayer : bool
- currentMode : GameMode
- currentDifficulty : Difficulty
+ Game()
+ start()
- randomizeShips(Board*, bool) : void
- playerAttack(int) : ShotResult
- AIAttack(int) : ShotResult
- addShipRandomly(string, Board*) : void
- addShipFromPlayer(string, Board*) : void
- printLastTurnResults(vector<ShotResult>) : void
- printCurrentTurnResults(vector<ShotResult>) : void
- readShips(Board*) : void
- run : void

Input Requirements

From user:		
Description	Data type	Range or valid input
Game difficulty	int	1 to 4
Attack coordinates	int/char	1 to 10/ ‘A’ to ‘J’
Game mode	int	1 to 2
One or two players	int	1 to 2
Player name	string	n/a
Option to play again	string	“Yes” “No” Not case sensitive

From file:		
Description	Data type	Range or valid input
Type of ship	string	“Carrier” “Battleship” “Cruiser” “Submarine” “Destroyer” Not case sensitive
X position	int	1 to 10
Y position	int	1 to 10
Orientation	string	“Horizontal” “Vertical” Not case sensitive

Output Requirements

The program will first ask the user to select a game mode, with two options: classic and salvo.

Then the program will ask how many people are playing.

If one person is playing, it will ask the user to choose positions for their ships if there are any errors in the ship placement file.

If two people are playing, it will ask each one if they want to place ships themselves or have the program generate positions randomly.

Whenever it's the user's turn, the game will output the enemy's last shot and the results of that shot, then it will print the planning grid and ship grid, then the results of the user's turn.

When someone wins, it will congratulate them, then display how many turns it took for them to win.

Problem Solution Discussion

When deciding where the AI player will place its next shot, the program first takes into account the game difficulty.

If the selected game difficulty is Easy, the AI chooses a random shot position (that hasn't been tried already).

If the difficulty is Normal, the AI has a 10% chance of choosing a space occupied by a player's ship, and an 90% chance of choosing a shot randomly.

If the difficulty is Hard, the AI will have a 15% chance of choosing a occupied by a player's ship, and a 85% chance of choosing a shot randomly.

If the difficulty is EXTREME, the AI will have a 20% chance of hitting a player's ship, and an 80& chance of choosing a random position.

Data Structures

I've considered using simple vectors over my Grid class because the vectors would have less dependency. But the Grid class makes it simple to print hit and miss locations, locations that haven't been tried yet, and easily setting values in the grid.

I've also considered expanding the Grid class to include Board functions, but I felt that it would complicated the Board class. It would be easier to derive the meaning by looking at the Board class with the two classes separate than it would be if the Board class had the variables and functions of a Grid class.

User Interface Scheme

No user interface