BattleShip Game

CIS 137 Final project

Game Overview

- Single-player Battleship game developed using SwiftUI
- The player selects a difficulty level (Easy or Hard) before starting
- Each difficulty controls the maximum number of shots allowed
- The game is played on an 8×8 grid
- Ships are placed randomly every time a new game starts
- The player taps on grid cells to fire shots
- The game tracks hits, misses, and the number of shots used
- The goal is to sink all ships before running out of shots
- The app includes animations, a reset option, and a custom home screen design

MVVM in the Battleship App

- The Battleship app is built using the **MVVM architecture**
- MVVM stands for **Model, View, and ViewModel**
- This structure separates the game logic from the user interface
- It helps keep the project **organized, clean, and easier to maintain**
- The **Model** stores the game rules and data
- The **ViewModel** controls the game state and difficulty
- The **Views** display the game and respond to user actions
- MVVM allows the UI to automatically update when the game state changes

Model (Model.swift)

The Model contains all core game logic and data structures:

- BattleshipGame – manages the grid, ship placement, and game rules
- CellState – represents each cell’s state (unknown, hit, miss)
- Coordinate – stores positions on the grid
- HitResult – represents the result of a shot

The Model does **not** contain any UI code.

ViewModel (BattleshipViewModel.swift)

The ViewModel:

- Stores the current game instance
- Tracks shots fired and hits
- Controls the maximum number of shots based on difficulty
- Updates the game status message
- Handles user taps on the grid
- Resets the game when needed

It acts as a bridge between the Model and the Views.

Views (ContentView.swift & GameView.swift)

The Views control the user interface:

- **ContentView**
  - Home screen
  - Displays the title
  - Lets the user choose a difficulty
  - Navigates to the game screen
- **GameView**
  - Displays the game board
  - Shows the status message and counters
  - Contains the Reset button
- **CellView (inside GameView)**
  - Draws each individual grid cell
  - Changes color based on hit or miss

Challenges & Solutions

Challenges:

- Managing grid updates correctly
- Connecting the Model to the UI
- Adding animations without breaking functionality

Solutions:

- Used `@Published` properties in the ViewModel
- Used SwiftUI animations for buttons and cells
- Followed MVVM structure to keep logic clean