Introduction
Product description in a few sentences

+ Battleship make with pygame
+ Players place ships on a grid and take turns trying to guess where their opponent's ships are hidden. The goal is to sink all of the opponent's ships before they sink yours. It's a fun and strategic game of naval warfare that's easy to learn and play!

The main parts of the software / modules and their interconnections Description of the main parts of the application/system/service and their communication

1. Game Engine Module: This module serves as the core of the application, managing game logic, player interactions, and overall gameplay flow. It orchestrates the interaction between various components and modules.

2. Graphics Module: Responsible for rendering visual elements such as game boards, ships, grid cells, and graphical user interfaces (GUI). It communicates with the game engine to display game state changes and user inputs accurately.

3. Input Module: Handles user inputs from keyboard, mouse. It communicates user actions, such as ship placement, firing shots, or navigating menus, to the game engine for processing.

4. User Interface (UI) Module: Provides the graphical user interface for players to interact with the game. It includes menus, buttons, dialogs, and other UI elements. It communicates user inputs and UI events to the game engine and graphics module.

Description of main functionalities, description of services Can be conceived as use cases - just a verbal description, does not need to be diagrams.

1. Game Setup: Players can set up the game by placing their ships on the grid-based game board. They can choose the size and orientation of each ship, ensuring strategic positioning for battle.

2. Turn-Based Gameplay: The game proceeds in a turn-based fashion, with each player taking turns to guess the coordinates of their opponent's ships and attempt to sink them. Players receive feedback on hits and misses to inform their next moves.

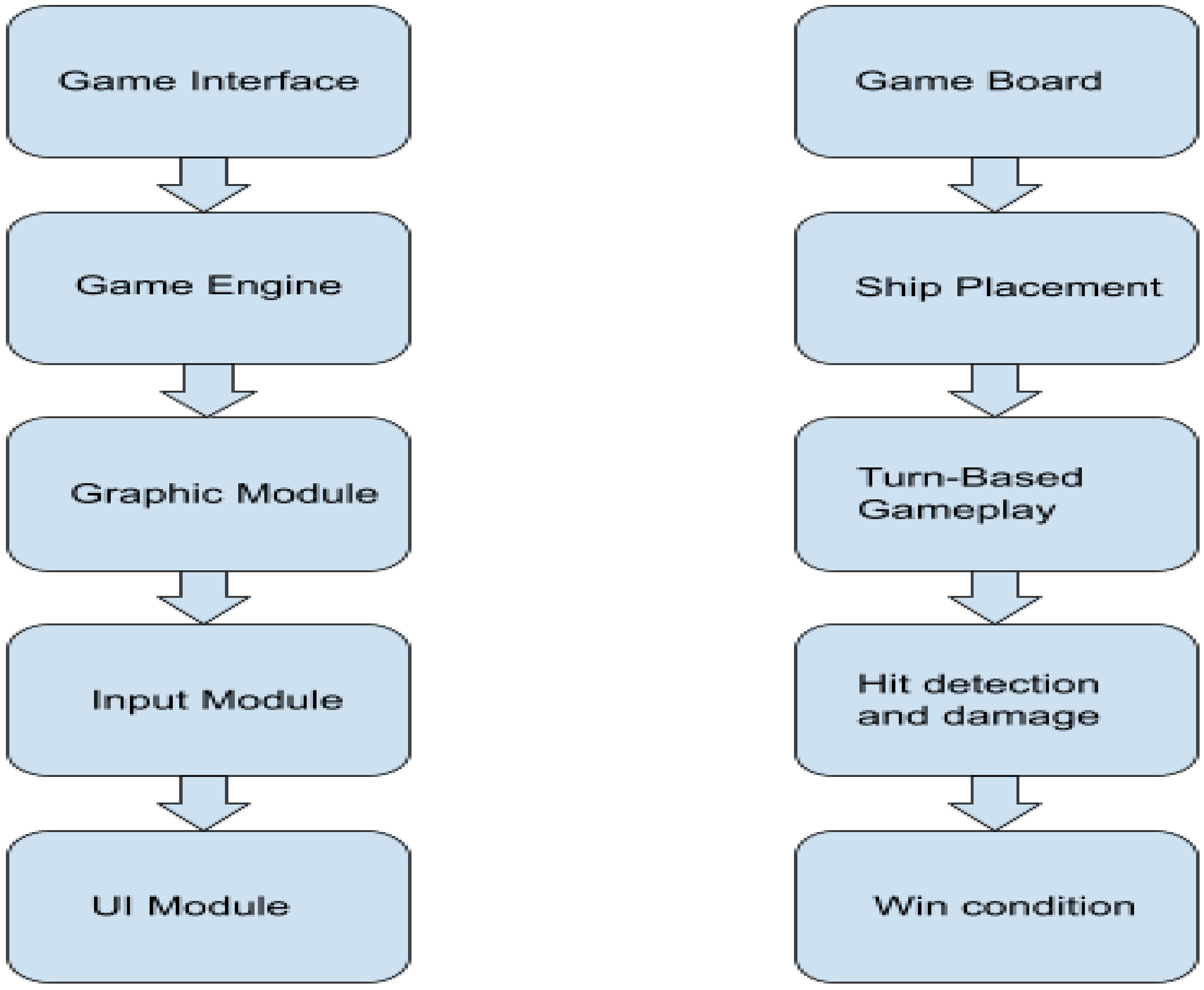
3. Ship Placement: Before the game begins, players can strategically position their ships on their own game board. They must carefully consider factors such as ship size, spacing, and concealment to maximize their chances of victory.

4. Targeting and Firing: During their turn, players select coordinates on the opponent's grid to fire shots and attempt to hit enemy ships. Successful hits result in damage to the targeted ship, while misses reveal empty ocean spaces.

5. Hit Detection and Damage: The game engine accurately detects hits and misses based on player inputs. When a shot lands on a ship's location, the game registers a hit and inflicts damage to the targeted ship. Ships are sunk when they sustain enough damage.

6. Win Condition: The game continues until one player successfully sinks all of their opponent's ships. The victorious player is declared the winner, and the game concludes with a congratulatory message.

Architecture sketch A simple sketch of the solution, showing client - server, backend - frontend, whether there is a database, possibly third party integration etc.



The Game Interface represents the entry point for players to interact with the game. **The Game Engine** orchestrates the game logic, managing player interactions, game state, and overall gameplay flow. **The Graphics Module** is responsible for rendering visual elements such as game boards, ships, and GUI components. The Input Module handles user inputs from keyboard, mouse, or other input devices. **The User Interface Module** provides the graphical user interface for players to interact with the game, including menus, buttons, and dialogs.	**The Game Board module** represents the grid-based game board where players place their ships and track their opponent's shots. **The Ship Placement module** handles the initial placement of ships on the game board before gameplay begins. **The Turn-Based Gameplay module** orchestrates the flow of the game, allowing players to take turns guessing coordinates and firing shots at their opponent's ships. **The Hit Detection and Damage module** detects hits and misses on the game board and inflicts damage to ships accordingly. **The Win Condition module** determines when a player has won the game by sinking all of their opponent's ships.

Error handling Http statuses for APIs, exception handling for other applications, etc.

Error Handling in Grid class:

Invalid Ship Placement:

Error handling implemented to prevent invalid ship placements, such as ships overlapping or extending beyond the grid boundaries.

The **isValidRowCol** method currently checks for some conditions related to ship placement validity

Ship Counting:

The countShipNotSunk and countShipsByLength methods provide counts of ships

Error handling ensure that ship counting functions correctly even if ships are not properly initialized or if their status is inconsistent.

Collision Detection:

The isCollideOtherShip method checks for collisions between ships

Error handling check for empty ship collections or verifying ship initialization before performing collision detection.

Error Handling in PlaceShipsScreen Class:

In the update method, there's conditional logic to handle the situation where the ship buttons need to be shown or hidden based on user interaction. This logic ensures that the button text is correctly set based on the state of ship visibility.

Error Handling in PlaceShipsScreen Methods:

In the grid1Update and grid2Update methods, there are several checks to handle potential errors:

Handling mouse input to update ship positions within the grid, ensuring that ships are moved or rotated correctly without causing collisions.

Properly updating ship positions based on mouse input and grid cell positions, considering ship boundaries and collision detection with other ships.

Error handling for creating new ships within the grid, ensuring that ships are instantiated correctly without overlapping existing ships.

Error Handling in MainGame Class:

In the update method, error handling is applied to determine the game over condition and the winning player. This logic ensures that the game state is updated correctly based on ship counts.

Similar error handling is applied in the draw method to display appropriate messages and handle the end of the game.

Error Handling in MainGame Methods:

Error handling is applied in the player1Turn and player2Turn methods to handle player turns and cell selections during gameplay. This includes checks to ensure valid cell selections and proper handling of hits and misses.

Overall, the error handling mechanisms in the Battleship game code focus on preventing issues such as ship collisions, invalid input, and game state inconsistencies to ensure smooth gameplay and accurate representation of the game state.

Environment Where it is deployed - production, test, sandbox ...

- Development Environment: window, pycharm
- Testing Environment: window, pycharm
- Deployed:

  Pygame applications can be deployed on various platforms, including desktop operating systems like Windows, macOS, and Linux.

  Windows: Pygame applications can be packaged as standalone executables (.exe files) using tools like PyInstaller or cx_Freeze. These executables can be distributed to users running Windows.

Technologies used

Python:

Language used for the entire project.

Version: Python 3.x (e.g., Python 3.7, Python 3.8).

Pygame:

Library for creating 2D games in Python.

Provides functionality for graphics, sound, input handling, and more.

Version: Pygame 2.x (e.g., Pygame 2.0.1).

NumPy:

Library for numerical computing in Python.

Used for efficient handling of multi-dimensional arrays (not actively used in the provided code but commonly used in game development).

Version: NumPy 1.x (e.g., NumPy 1.19.5).

Resources Documentation of the technologies and libraries used, third party documentation that I use in the project, other technical documentation that I have used - e.g. for git, for the cloud solution where the application is deployed, etc.

Python Documentation:

Official Python documentation, providing information about language syntax, standard libraries, and built-in functions.

Link: https://docs.python.org/3/

Pygame Documentation:

Official documentation for Pygame library, providing detailed explanations of modules, classes, functions, and examples.

Link: https://www.pygame.org/docs/

NumPy Documentation:

Official documentation for NumPy library, offering comprehensive guides, tutorials, and references for array manipulation and numerical operations.

Link: https://numpy.org/doc/

Aseprite Documentation:

The official Aseprite documentation provides comprehensive guides, tutorials, and references for using the software's features and tools.

It covers topics such as creating pixel art, animation, layers, brushes, palettes, and exporting assets.

Link: https://www.aseprite.org/docs/

Dafont:

For download fonts

Link: https://www.dafont.com/