1. Overview:
    a. The following project was inherited by Group 23 from Group (). We were tasked with refactoring the existing repository after forking it. The project is coded in python and uses the pygame library for the majority of its functionality. Currently, the project source code is a singular file composed of large functions that handle all the functionality of the game. To refactor the code, we will take these existing functions and reorganize them into classes to allow for a more modular approach. The benefits of a modular approach allow for object oriented programming, more easily allowing for the AI implementation through an extension of the Player Class.
2. Installation and Setup:
    a. The basic requirements to run the application on a specified machine include the machine having the latest version of python installed as well as the pygame library installed.
        i. Commands to run:
            1. Git clone <repo url>
            2. Brew install python
            3. Pip install pygame
3. Game Rules:
    a. Ship placement
        i. Old implementation:
            1. Each player defaults to five ships being placed. This is hardcoded and cannot be changed. You do have the option between vertical and horizontal placement, and ship placement must NOT overlap
        ii. Latest Implementation:
            1. This implementation keeps the same features as the old implementation, but now players have the option to select 1-5 ships to play with.

b. Battle phase

    i. Old implementation:

        1. Two players in the same room play against one another, passing the computer back and forth between turns firing at a blank opponent grid guessing where their opponent's ships are placed. Every shot returns a message to know whether it was a hit or miss.

    ii. Latest implementation

        1. Players have the option to play the same way as the old implementation or they can play against an AI with three difficulty options: easy, medium, or hard. Easy mode has the AI shooting at random every turn. Medium mode has the AI shooting at random until a hit is registered, in which case it fires adjacently for subsequent turns until a ship is sunk. Hard mode has the AI targeting a ship-occupying square on every turn

4. Code Structure

a. Original

    i. Function based code structure

    ii. One python file

b. Updated:

    i. Object oriented programming→ refactored to use classes for functions that could be methods of the same class

    ii. Modular file structure

        1. battle_screen.py

        2. pass_screen.py

        3. button.py

        4. placement_screen.py

5. player.py
6. script.py
7. start_screen.py
8. win_screen.py

5. Classes and Functions

a. player.py

i. No functions

1. Parameters
    a. Player ID (identify whether player 1 or 2)
2. Returns
    a. Stores players' ships, hits, sunk ships, whether special shot has been used

b. ai.py

i. level_one(self, player)

1. Parameters
    a. Player (to be used as AI)
2. Returns
    a. Returns behavior of easy mode AI. When fire is called with this difficulty, a random square is fired upon every time

ii. level_two(self, player)

1. Parameters
    a. Player (to be used as AI)
2. Returns
    a. Returns behavior of medium level AI. When fire is called with this difficulty, a random square is fired upon every time until a

hit is registered. From there, adjacent squares are fired at until the ship is sunk

iii. level_three(self, player)

1. Parameters
    a. Player (to be used as AI)
2. Returns
    a. Returns behavior of hard level AI. When fire is called with this difficulty, a square occupied by a ship is fired at on every turn

iv. fire(self, player)

1. Parameters
    a. player
2. Returns
    a. Uses levels 1-3 to determine how squares are fired at each turn by the AI

c. button.py

i. draw(self)

1. Parameters: None
2. Returns
    a. Draws button on display to be interacted with by the user

ii. click(self, mouse_pos)

1. Parameters
    a. Mouse position
2. Returns
    a. Check if button is clicked and mouse is in correct position for button to register as clicked

d. script.py

i. main()

1. Parameters: None
2. Returns
    a. Controls flow of the game initiating instances of all game states and displaying a loop of battle and pass screens until a winner is declared

e. battle_screen.py

i. draw_grid(self, gridParams, opponent=False)

1. Parameters
    a. Parameters of the grid (x and y dimensions, player 1 or 2), opponent (false if none)
2. Returns
    a. Displays grid with stored hits, misses, ships along with grid and cell borders

ii. handle_special(self, event, playerGridParams, opponentGridParams)

1. Parameters
    a. Event type, player's and opponent's and grid parameters stored in the dictionaries
2. Returns
    a. Handler for when player fires 3x3 special shot

iii. handle_attack(self, event, playerGridParams, opponentGridParams)

1. Parameters
    a. Event type, player's and opponent's and grid parameters stored in the dictionaries
2. Returns

a. Handler for when player fires a shot, storing if a ship is hit and returning whether ship was hit, missed, sunk, or already attacked

iv. get_ship(self, opponent, x, y)

1. Parameters
    a. Opponent, x and y coordinates
2. Returns
    a. Finds ship at given coordinates

v. check_ship_sunk(self, player, ship)

1. Parameters
    a. Player 1 or 2, ship specified
2. Returns
    a. Checks to see whether all coordinates belonging to a single ship have been hit

vi. all_ships_sunk(self, player)

1. Parameters
    a. Player 1 or 2
2. Returns
    a. Checks if all ships are sunk belonging to the player

vii. end_turn(self)

1. Parameters: None
2. Returns
    a. Signals that the player's turn is finished

viii. toggle_special(self)

1. Parameters: None
2. Returns
    a. Toggles the use of the special 3x3 shot

ix. display(self, player)

1. Parameters
    a. Player 1 or 2
2. Returns
    a. Displays all info going on in the battlescreen on the pygame window

f. Pass_screen.py

i. display(self)

1. Parameters
    a. Player (1 or 2)
2. Returns
    a. Displays screen to tell player to pass the display to their opponent so that the opponent can take their turn before passing the display back to the player (maintains privacy of players' boards)

g. placement_screen.py

i. display(self, player)

1. Parameters
    a. Player (1 or 2)
2. Returns
    a. Displays screen for placing ships on the board calling on the other helper functions to create icons for placing ships

ii. _draw_text(self, text, pos)

1. Parameters
    a. Text to be displayed, position on the screen
2. Returns

a. Helper function to help render and display text on the screen for "Player (1 or 2) Ship Placement"

iii. _draw_grid(self, grid)

1. Parameters
    a. Grid
2. Returns
    a. Helper function to draw the placement grid on which ships are placed

iv. _draw_ships(self, ships)

1. Parameters
    a. Ships
2. Returns
    a. Helper function to draw ships on the side of the screen for placement on the grid

v. _draw_placement_indicator(self, ship, mouse_pos, vertical)

1. Parameters
    a. Ship, mouse position, whether ship is vertical/horizontal
2. Returns
    a. Helper function to display a placement indicator for a given ship before it is placed on the board

vi. _toggle_orientation(self)

1. Parameters: None
2. Returns
    a. Changes orientation of ship between vertical and horizontal

vii. _finish_placement(self, all_ships_placed)

1. Parameters

a. All_ships_placed (check to see if all ships are placed)

2. Returns

a. Determines whether ship setup is finished

viii. _handle_events(self, ships, grid)

1. Parameters

a. Ships, grid

2. Returns

a. Handles user input events

ix. _handle_mouse_click(self, event, ships, grid)

1. Parameters

a. Event, ships, grid

2. Returns

a. Handles mouse clicks for placement of ships and selection

x. clear_ship(self, ship_num, grid)

1. Parameters

a. Ship number, grid

2. Returns

a. Removes a ship from the grid

xi. is_valid_placement(x, y, size, is_vertical, ship_num, grid)

1. Parameters

a. Dimensions and size of ship, orientation of ship, ship number, grid

2. Returns

a. A function to check if placement of the ship in a specific location is valid

xii. _store_ships(self, player, grid)

1. Parameters
    a. Player (1 or 2), grid
2. Returns
    a. Stores placement of ships on grid within the game parameters

h. Start_screen.py

i. display(self)

1. Parameters: None
2. Returns
    a. Sets up display screen for starting menu calling on other functions to prompt user for whether they want to play against a player or AI, what level of difficulty the AI should be, and how many ships to play with

ii. display_mode_selection(self)

1. Parameters: None
2. Returns
    a. Set up display for choosing PvP or PvAI. 2 buttons for “Player” or “AI”

iii. select_player_mode(self)

1. Parameters: None
2. Returns
    a. Establishes that the player has selected to play against another person. Stores this information in booleans so that the loop within display moves the player to the next phase (choosing ship number)

iv. select_ai_mode(self)

1. Parameters: None

2. Returns
    a. Establishes that the player has selected to play against AI. Stores this information in booleans so that the loop within display moves the player to the next phase (choosing AI difficulty)

v. display_ai_difficulty_selection(self)
    1. Parameters: None
    2. Returns
        a. Displays screen with 3 buttons to select AI difficulty: easy, medium, or hard

vi. set_difficulty(self, difficulty)
    1. Parameters
        a. Difficulty
    2. Returns
        a. Passes information from buttons pressed within display_ai_difficulty_selection() to set the difficulty level of the AI

vii. display_ship_selection(self)
    1. Parameters: None
    2. Returns
        a. Displays screen with 5 buttons to determine number of ships to play with (1-5 ships)

viii. set_num_ships(self, num)
    1. Parameters
        a. Number of ships to play with
    2. Returns
        a. Sets number of ships for the game

i. win_screen.py

i. new_game(self)

1. Parameters: none
2. Returns
    a. Starts a new game of battleship and resets all previous variables and states

ii. end_game(self)

1. Parameters: None
2. Returns
    a. Ends the game and shuts down the program

iii. display(self, player)

1. Parameters
    a. Player (1 or 2)
2. Returns
    a. Displays game over screen with which player was victorious and presents 2 buttons to either restart the game or exit the game

6. Controls

a. Start screen
    i. Two buttons to select whether to play another local player or an AI
b. Difficulty selection screen
    i. If AI is selected previously, the option to select an AI difficulty is presented with three buttons for easy, medium, and hard modes
c. Ship number screen
    i. Five buttons to select between one and five ships to play with
d. Ship placement screen

i. Depending on how many ships were selected previously, different sized ships will be available to place by clicking and dropping the ship on a desired space
ii. A button to rotate the ships so that they are horizontal/vertical is present
iii. A button to finish this phase is available. This will take player 1 to the battle screen

e. Battle screen
    i. Your board with ships and opponent's hits/misses is displayed on the left
    ii. A copy of the opponent's board with stored hits and misses from previous turns is displayed on the right. You may press any tile on the opponent's board to fire a shot at that square.
    iii. The option to fire a 3x3 special shot is a button in the corner which fires in all squares bordering the clicked square. Only one of these is permitted per game.
    iv. A button to finish turn and move to the pass screen is next to the special move button
f. Pass screen
    i. A button to finish passing is the only button present. This screen is simply to prevent players from seeing opponent's boards between turns
g. Win screen
    i. Two buttons to restart or exit the game are present

7. Authors

a. John Mosely
b. Paul Dykes
c. Aryamann Zutshi
d. Will Battey
e. Spencer Addis

8. References

a. Chat GPT

   i. Understanding lambda functions

b. Stack Overflow

Estimate to complete Project 2: 15 hours

Date	Description	Time to complete	Author(s)
09/24/2024	Met as group to discuss plan of action for completing requirements. Created UML diagram to reorganize previous code	2 hours	John, Willem, Spencer, Aryamann, Paul
09/27/2024	● Met to discuss revisions John and Paul made to their code. They created new classes for all the major things in the game. Battle screen, button, etc. Worked on the implementation of our special addition which is a 3x3 bomb ● Assigned individual files to each member, everyone has their own branch using their name, will share one test branch to push to, shouldn't affect anyone's own work since no file is being worked on by two individuals	2 hours	Willem, John, Spencer, Aryamann, Paul
09/28/2024	Refactored previous code into functions that can be used during the battle phase of the game	3 hours	John
09/28/2024	Created start screen so that AI options can be displayed in beginning of the game	2 hours	Spencer
09/28/2024	Worked on win screen to refactor	2 hours	Willem

	previous code into final screen with options to restart or exit		
09/28/2024	Alter placement screen so that ships can be placed 1-5 depending on user input using functions	2 hours	Aryamann
09/28/2024	Button class created; simplify creating buttons for work	2.5 hours	Paul
09/29	Met virtually to show what we have all been working on and put it all together. Code refactoring got done and are now working on the implementation of the AI and 3x3 bomb. The AI logic was already completed on Saturday.	10 hours	Willem, John, Spencer, Aryamann, Paul

TOTAL: 25.5 hours