Project 1 – Battleship Game

Team 17 - Brisa Andrade, Andy Trinh, Muskan Sharma, Jennifer Aber, Abraham Lopez

Overview

Project 1 is a Battleship game that can be played on two devices, with one player on each device. Ships are placed on a 10x10 grid, and players can guess the location of their opponent's ships. Each player makes a series of attacks, and once all grid locations containing a ship have been successfully targeted, the player who makes the final attack wins the game. This version of Battleship is played over a client-server connection.

How the Game Works

Battleship can run on Windows or Linux machines if they support Python. The Python gaming library (PyGame) will need to be installed as well. Once these conditions have been met, Battleship can be launched from a command line by typing the following command:

```
python (or python3) main.py
```

This command launches the setup screen where you can either host a game or join an existing game. Both players will need to be on the same local network. You can enter a username in the text box and Begin Battle.

Once a connection to the other player has been established, you can place your ships by using click and drag. There are five ships, and you can place them at any location on the grid. To rotate the ship vertically, press the left arrow key. Squares on the grid are highlighted, indicating ship placement at that location. After placing all your ships, the Start Game button begins the game.

Gameplay consists of two main screens: the grid containing your placed ships, and the grid showing the locations of your attacks. Players continue taking turns attacking the other's grid until all highlighted locations have been attacked, indicating an opponent's ships have been destroyed.

Code Structure

Battleship code consists of seven main modules: main.py, menu.py, client.py, server.py, battle.py, setup.py, and pygame_textinput.py. The application's structure and description of each module are included below. Comments included in the code itself provide a more detailed, line-by-line explanation of code functionality.

main.py is the main driver for the Battleship program. The flow of the game, including the menu screen, the client/server interfaces, and the board are all called in main. These gameplay screens are structured as an array of objects – one of each class type (Menu, Server, Client, BoardSetup). Two functions – get_ip() and get_username() allow the user to enter the ip address of a host to join and a username.

The flow of the game is determined by a series of if-else statements that utilize built-in pygame events triggered by user input, such as `MOUSEBUTTONDOWN` and `QUIT`. Depending on the location on the screen where the event occurs (outlined by x and y coordinates), you can move between screens as you enter a username, join a session, host a game session, place ships, and attack the other player's board. This functionality is driven by a series of Boolean variables, which are changed as the player navigates the different options.

setup.py handles the setup of the game grid. This module starts by declaring the BoardSetup class, which includes seven class functions:

- `init`: initializes an object of the BoardSetup class
- `generateGrid(self)`: generates a 10x10 grid board
- `shipSet()`: places ships on the board
- `click(self, image, event)`: loads the ship images to the mouse to allow the user to click and drag the images onto the board
- `generatePos(self)`: generates grid values for the game board
- `calcPos(self, ship)`: calculates the position of the ship on the board and marking squares as occupied by a ship
- `run_first(self)`: handles the initial setup of the game board, calling the `generateGrid()` and `shipSet()` functions

menu.py implements the class menu and creates the initial Welcome screen. This module displays the initial *Battleship* title and gives the user the option to Join Party or Create Party. This module contains four main subroutines:

- `heading(self)`: initializes title - *Battleship*
- `joinButton(self)`: loads file `join.png`

- `createButton(self)`: loads file `create.png`
- `run(self)`: calls three functions above and generates the overall display

battle.py handles the back-and-forth gameplay, which includes attacking an opponent's grid and updating each player's grid with hits and misses. It establishes a network stream so the players can communicate with each other, then handles the turn taking aspect of the game. Players will make their guesses on a 10x10 grid and will find out if their attack was a hit or a miss. Another 10x10 grid keeps track of their guesses and marks locations of previous hits and misses. If an opponent makes a successful attack and hits one of their battleships, a player's ships grid will be updated as well. This module contains the following functions:

- `randomBoard(self)`: generates a 10x10 grid to place attacks
- `heading(self)`: establishes title and text in the game board
- `createSocket(self, ip)`: establishes a network stream for simultaneous gameplay
- `genMyBoard(self)`: generates player board for gameplay
- `genEnemyBoard(self)`: generates enemy board
- `attempt(self)`: sets up text for attack notification
- `switchBoard(self, x, y)`: switches between boards during battle
- `message(self, desc)`: generates text showing hits and misses
- `run(self, ip)`: driver function that handles workflow of this module

pygame_textinput.py handles the text input portion of this game. This module is part of the PyGame library. For more information, a helpful link is included here: https://github.com/Nearoo/pygame-text-input

client.py permits a second player to join the game and interact with the game initialized in the server class. Functions included are:

- `heading(self)`: generates header message: "Waiting for game to start..."
- `background(self)`: fills screen with a background color
- `conClient(self, host, username)`: connects to the host server
- `decrypt(self, ip)`: decrypts the invite code sent through the server module
- `run(self, username, ip, first)`: implements client functionality

server.py establishes a game server for players to connect. It also provides graphics indicating 'username' versus 'enemyUsername". It includes the following functions:

- `createServer(self)`: establishes a server for clients to connect to the game
- `getInvite(self)`: generates a team invite code (IP address) to connect to the multiplayer game
- `username(self,name)`: prints username
- `genVers(self)`: generates the text “Versus”
- `enemyUsername(self)`: displays enemy username
- `beginBattle(self)`: implements the start of the game
- `run(self, username, first, second, changeLoc, sendShip)`: implements the server functionality