INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

-----*-----**

PROJECT REPORT

Net - centric

Net - centric Programming (IT076IU)

Semester 2 - The academic year 2023 - 2024

TABLE OF CONTENTS

TABLE OF CONTENTS....2
CONTRIBUTION TABLE....4
1. Project Description....5
2. Pokedex....5
Introduction....5
Project Structure....5
Data Structures....5
Constants....5
Variables....6
Implementation....6
Main Function....6
crawlPokemonsDriver Function....6
crawlPokemons Function....6
Usage....6
Challenges and Solutions....7
Challenge: Handling Dynamic Web Pages....7
Challenge: Parsing Various Data Formats....7
3. PokeBat....7
Introduction....7
Objectives....7
Project Structure....7
Directories....7
Data Structures....7
Constants....8
Implementation....8
Server (server/server.go)....8
Client (client/client.go)....8
Usage....9
Running the Server....9
Running the Client....9
Challenges and Solutions....9
Challenge: Handling Multiple Players....9
Challenge: Authenticating Users....9
Challenge: Managing Game State....9
Challenge: Ensuring Real-time Communication....9
4. PokeCat....10

Introduction..... 10
Objectives..... 10
Project Structure..... 10
Directories..... 10
Data Structures..... 10
Variables..... 10
Implementation..... 11
Server (pokecat/pokecat.go)..... 11
Client (client/client.go)..... 11
Usage..... 11
Running the Server..... 11
Running the Client..... 11
Challenges and Solutions..... 12
Challenge: Handling Concurrent Players..... 12
Challenge: Managing Game State..... 12
Challenge: UDP Communication..... 12

CONTRIBUTION TABLE

No.	Full Name	Student's ID	Tasks
1	Phạm Vũ Quang	ITITIU21096	Creating testing unit, making CI/CD, deploying server for backend, creating endpoints.
2	Nguyễn Tiến Hưng	ITITDK21072	Making the main page, creating stages & project's page structure.

1. Project Description

This report covers three distinct projects that leverage Go (Golang) to create interactive applications involving Pokémon data. Each project addresses different aspects of interaction with Pokémon data, ranging from web scraping to multiplayer gaming.

The primary goal of these projects is to demonstrate the versatility and capability of Go in handling various tasks, including web scraping, concurrent programming, and real-time multiplayer interactions.

2. Pokedex

Introduction

The project involves developing a web scraper to extract detailed information about Pokémon from an online Pokédex. The extracted data is then stored in a structured JSON format. The scraper navigates through the website, extracts various attributes for each Pokémon, and saves the data in a file for further use.

It is done using the Playwright-go Library, the library can be found here: link

Project Structure

Data Structures

The project defines several Go structs to model the data for each Pokémon:

- **Stats**: Holds statistics like HP, Attack, Defense, etc.
- **GenderRatio**: Represents the male and female ratios.
- **Profile**: Contains attributes like height, weight, catch rate, etc.
- **DamegeWhenAttacked**: Stores elements and their damage coefficients when the Pokémon is attacked.
- **Moves**: Details about the moves a Pokémon can learn.
- **Pokemon**: The main struct that aggregates all the above information.

Constants

- `numberOfPokemons`: The number of Pokémon to scrape.
- `baseURL`: The base URL of the website to scrape.

Variables

- `pokemons`: A slice to store the scraped Pokémon data.

Implementation

Main Function

The `main` function initiates the scraping process by calling the `crawlPokemonsDriver` function with the number of Pokémon to scrape.

crawlPokemonsDriver Function

1. Initializes Playwright and launches a Chromium browser instance.
2. Navigates to the base URL.
3. Iterates through the specified number of Pokémon.
4. For each Pokémon, simulates a button click to open its details and calls `crawlPokemons` to scrape the data.
5. After scraping, converts the collected data to JSON and writes it to a file.
6. Closes the browser and stops Playwright.

crawlPokemons Function

This function extracts detailed information for a single Pokémon:

1. **Stats**: Extracts statistics like HP, Attack, Defense, etc.
2. **Profile**: Extracts profile details including height, weight, catch rate, etc.
3. **Gender Ratio**: Extracts the male and female ratio.
4. **Damage When Attacked**: Extracts the elements and their damage coefficients.
5. **Evolution**: Extracts evolution details.
6. **Moves**: Extracts details about the moves a Pokémon can learn.
7. **Elements**: Extracts the Pokémon's elements (types).

Finally, the scraped data for the Pokémon is appended to the `pokemons` slice.

Usage

1. Install the Playwright-Go library.
2. Move to `crawler` directory with `cd crawler`
3. Run the script using `go run crawler.go`
4. The script will navigate through the pokedex.org and then crawl all of the data.

Challenges and Solutions

Challenge: Handling Dynamic Web Pages

- **Solution**: Used Playwright's capabilities to interact with dynamic web pages, including clicking buttons and waiting for elements to load.

Challenge: Parsing Various Data Formats

- **Solution**: Utilized Go's string manipulation and conversion functions to parse data accurately.

3. PokeBat

Introduction

The PokeBat project is a TCP-based multiplayer Pokémon battle game where players can authenticate, receive random Pokémon, and engage in battles. The server handles player authentication, assigns Pokémon, and facilitates battles. The client connects to the server, allows user interaction, and processes game instructions.

Objectives

1. Implement a TCP server to handle multiple players and facilitate Pokémon battles.
2. Allow players to authenticate and join the game.
3. Assign random Pokémon to players and manage battles.
4. Ensure smooth communication between the server and clients.

Project Structure

Directories

- **server/**: Contains the server-side code for handling game logic.
- **client/**: Contains the client-side code for player interactions.

Data Structures

The project defines several Go structs to model the game data:

- **User**: Represents a player with a username and password.
- **Stats**: Holds Pokémon statistics like HP, Attack, Defense, etc.

- **Profile**: Contains attributes like height, weight, catch rate, etc.
- **Damage**: Stores elements and their damage coefficients.
- **Pokemon**: The main struct that aggregates all the above information.
- **Player**: Represents a player with attributes like name, connection, Pokémon, and active Pokémon index.
- **Battle**: Represents a battle between two players.

Constants

- `HOST`: The host address for the server.
- `PORT`: The port on which the server listens.
- `TYPE`: The type of connection (TCP).
- `MIN_PLAYER`: Minimum number of players required to start a battle.
- `POKEDEX_FILE`: Path to the file containing Pokémon data.

Implementation

Server (server/server.go)

Main Function

1. Starts the TCP server and listens for incoming connections.
2. Authenticates players and adds them to the player list.
3. Loads Pokémon data from a file.
4. Assigns random Pokémon to players once the minimum player requirement is met.
5. Initiates a battle between the first two players.

Helper Functions

- `authenticate`: Verifies user credentials.
- `loadUsersFromFile`: Loads user data from a JSON file.
- `hashPassword`: Hashes the password using SHA-256.
- `loadPokemonFromFile`: Loads Pokémon data from a JSON file.
- `getRandomPokemons`: Shuffles and selects a subset of Pokémon.
- `chooseStartingPokemon`: Prompts the player to choose their starting Pokémon.
- `runBattle`: Manages the battle logic, including player turns and actions.
- `readFromConn`: Reads data from a connection.
- `performAttack`: Handles the attack logic during a battle.
- `switchPokemon`: Allows a player to switch their active Pokémon.
- `endBattle`: Ends the battle and notifies the players of the result.

Client (client/client.go)

1. Connects to the server using TCP.
2. Prompts the user for authentication details.
3. Sends authentication data to the server and processes the response.
4. Listens for game instructions from the server and sends user inputs.
5. Displays game messages and handles game flow based on server instructions.

Usage

Running the Server

1. Ensure Go is installed on your system.
2. Navigate to the `server` directory.
3. Run the server using `go run server.go`

Running the Client

1. Navigate to the `client` directory.
2. Run the client using `go run client.go`
3. Enter your username and password to authenticate and join the game.

Challenges and Solutions

Challenge: Handling Multiple Players

- **Solution**: Used Go's `net` package to handle multiple TCP connections and manage player interactions concurrently.

Challenge: Authenticating Users

- **Solution**: Implemented a simple authentication mechanism using username and password, with password hashing for security.

Challenge: Managing Game State

- **Solution**: Maintained separate structs for players and battles to keep track of the game state and ensure smooth transitions between actions.

Challenge: Ensuring Real-time Communication

- **Solution**: Used blocking reads and writes on TCP connections to facilitate real-time communication between the server and clients.

4. PokeCat

Introduction

The project involves developing a multiplayer Pokémon game using UDP for communication. The game allows multiple players to connect to a server, move around a virtual world, encounter Pokémon, and manage their inventory. The server handles player interactions, Pokémon encounters, and maintains the state of the game world.

Objectives

1. Implement a multiplayer game using UDP for communication.
2. Allow players to move around a 20x20 grid world.
3. Simulate Pokémon encounters and allow players to catch Pokémon.
4. Store player information and inventory in JSON files.

Project Structure

Directories

- **pokecat/**: Contains the server-side code for handling game logic.
- **client/**: Contains the client-side code for player interactions.

Data Structures

The project defines several Go structs to model the game data:

- **Stats**: Holds Pokémon statistics like HP, Attack, Defense, etc.
- **GenderRatio**: Represents the male and female ratios.
- **Profile**: Contains attributes like height, weight, catch rate, etc.
- **DamegeWhenAttacked**: Stores elements and their damage coefficients when the Pokémon is attacked.
- **Moves**: Details about the moves a Pokémon can learn.
- **Pokemon**: The main struct that aggregates all the above information.
- **Pokedex**: Contains a list of Pokémon and their coordinates in the game world.
- **Player**: Represents a player with attributes like name, ID, coordinates, inventory, and UDP address.

Variables

- `players`: A map to store active players.

- `pokeDexWorld`: A map to store Pokémon in the game world.
- `PokeWorld`: A 20x20 grid representing the game world.

Implementation

Server (pokecat/pokecat.go)

Main Function

1. Initializes the game world by placing random Pokémon.
2. Sets up the UDP server to listen for incoming connections and commands.

Helper Functions

- `randomInt`: Generates a random integer.
- `passingPokemonIntoInventory`: Adds a Pokémon to a player's inventory.
- `passingPlayerToJson`: Saves player data to a JSON file.
- `getRandomPokemon`: Retrieves a random Pokémon from the Pokédex.
- `positionOfPokemon`: Assigns random coordinates to a Pokémon.
- `checkForPokemonEncounter`: Checks if a player encounters a Pokémon.
- `printWorld`: Generates a string representation of the game world.
- `PokeCat`: Places a player in the game world and handles encounters.
- `movePlayer`: Moves a player in the specified direction.

Client (client/client.go)

1. Prompts the user to enter a username.
2. Connects to the server using UDP.
3. Sends commands to the server based on user input.
4. Listens for responses from the server and displays them.

Usage

Running the Server

1. Ensure Go is installed on your system.
2. Navigate to the `pokecat` directory.
3. Run the server using `go run pokecat.go`

Running the Client

1. Navigate to the `client` directory.

2. Run the client using `go run client.go`
3. Enter your username and start interacting with the game world using commands.

Challenges and Solutions

Challenge: Handling Concurrent Players

- **Solution**: Used Go's `synx.Mutex` to ensure thread-safe operations on player data.

Challenge: Managing Game State

- **Solution**: Maintained separate maps for players and Pokémon to keep track of the game state.

Challenge: UDP Communication

- **Solution**: Used Go's `net` package to handle UDP communication and manage incoming and outgoing messages efficiently.

Conclusion

The Pokémon projects presented in this report collectively showcase the power and versatility of Go (Golang) in handling diverse programming challenges. Each project demonstrates unique aspects of Go's capabilities, from web scraping to real-time multiplayer interactions, highlighting the language's efficiency, robustness, and suitability for various tasks.