VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NET-CENTRIC PROGRAMMING

PROJECT REPORT

Net - centric Programming (IT076IU)

Semester 1 - The academic year 2024 - 2025

TOPIC: TEXT-BASED POKEMON

Group: 15

Student's name	Student's ID	Contributions
Nguyễn Hoàng Minh Nghĩa	ITITIU21255	50%
Vũ Văn Đô	ITITIU21125	50%

1. Project Overview

This report includes three different projects that use Go (Golang) to create interactive applications involving Pokémon data. Each project focuses on a different element of interacting with Pokémon data, such as web scraping or multiplayer gaming.

The major purpose of these projects is to demonstrate Go's versatility and capability in a variety of tasks, such as web scraping, concurrent programming, and real-time multiplayer interactions.

2. Pokedex

Introduction

The PokeDex project is programmed to fetch the details of each pokemon on the www.pokedex.org website using a web crawler. The web crawler enters the web page of each pokemon, gets each attribute of the pokemon, and saves it in structured data. The retrieved data is then stored in JSON format with defined structures to be used in other projects.

Project Structure

Data Structures

- **Stats**: Contains information such as HP, Attack, Defense, etc. of each Pokemon.
- **GenderRatio**: Defines the ratio between male and female pokemons.
- **Profile**: Defines other information like height, weight, catch rate, etc
- **DamageWhenAttacked**: Holds damage multipliers depending on elements when the Pokemon is attacked.
- **Moves**: Contains information about the moves a Pokemon can learn.
- **Pokemon**: Major struct to hold the aforementioned structures.

Constants

- numberOfPokemons: Define how many Pokemons to crawl.
- baseURL: The URL of pokedex for the program to crawl.

Variables

- pokemons**:** A slice containing the crawled data of Pokemons.

Implementation

Main Function

The Main function simply calls the crawlPokemonDriver function with the given numberOfPokemons value to begin the crawling of the Pokedex website.

crawlPokemonsDriver Function

1. The function starts by initializing Playwright and launching a Chromium browser instance.

2. A new page is created, and it navigates to the base URL of the Pokémon database.
3. The function contains a loop that iterates through the range of Pokémon specified by numsOfPokemons.
4. For each Pokémon:
    - It constructs a locator string to find the button corresponding to the Pokémon.
    - It simulates a click on the button to open the Pokémon's details.
    - After a brief pause, it calls the crawlPokemons function to extract the details of the Pokémon.
5. After collecting data for all specified Pokémon, the function marshals the collected Pokémon data into a JSON format.
6. The JSON data is then written to a file named pokedex.json in the specified directory.
7. Finally, the function ensures that the browser instance is closed and Playwright is stopped to free up resources.

crawlPokemons Function

1. A Playwright Page object represents the web page containing the Pokémon details.
2. The function utilizes several structures to organize the data:
    - **Pokemon**: Represents the Pokémon and contains fields for its name, elements, stats, profile, damage when attacked, evolution level, next evolution, and moves.
    - **Stats, Profile, DamegeWhenAttacked, and Moves**: These structs are used to encapsulate specific attributes of the Pokémon.
3. The function extracts data from the web page using locators to find specific elements that contain the desired information:
    - **Stats Extraction**: It retrieves the Pokémon's HP, Attack, Defense, Speed, Special Attack, and Special Defense from the stats section of the page.
    - **Name Extraction**: The Pokémon's name is obtained from the header of the detail panel.
    - **Profile Extraction**: It gathers additional information such as height, weight, catch rate, gender ratio, egg group, hatch steps, and abilities.
    - **Damage When Attacked**: The function collects data on how the Pokémon is affected by different elements when attacked.
    - **Moves Extraction**: It retrieves the moves available to the Pokémon, including their names, elements, power, accuracy, PP (Power Points), and descriptions.
4. After extracting all relevant information, the function appends the populated Pokemon struct to a global slice named pokemons, which holds all the Pokémon data collected during the crawling process.

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 pokedex.org and then crawl all of the data.

Challenges and Solutions

Challenge: Handling Dynamic Web Pages

- Solution: Implement Playwright library to perform interactions with dynamic web pages, such as entering web pages and loading elements.

Challenge: Parsing Various Data Formats

- Solution: Parse multiple data formats using Go's string conversion functions.

3. PokeBat

Introduction

The PokeBat project is a multiplayer Pokémon-themed game that combines elements of exploration, battle, and Pokémon collection. It is designed to allow players to connect to a server, interact with the game world, encounter Pokémon, and engage in battles. The project is

structured using various components, including a server, client applications, and a data management system for Pokémon information.

Objectives

1. Manage players and Pokemon engagements by programming a TCP server.
2. Implement a simple authentication system with a username and password.
3. Allow players to have random Pokemons.
4. Handle stability between server and clients.

Project Structure

Directories

- **server/**: Include server program for managing interactions.
- **client/**: Include client program for player inputs and messages.
- **lib/**: Include data of Pokemons fetched from PokeDex and user credentials.

Data Structures

- **User**: Defines player structure with username and password.
- **Stats**: Contains details about a Pokemon like HP, Attack, Defense, etc.
- **Profile**: Holds other statistics like height, weight, catch rate, etc.
- **Damage**: Contains elements and damage multipliers.
- **Pokemon**: The main structure to hold the aforementioned information.
- **Player**: Defines a player structure containing attributes such as name, connection, Pokémon, and active Pokémon index.
- **Battle**: Defines 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)

The server code is responsible for managing player connections, game state, and interactions between players. It handles authentication, player movement, Pokémon encounters, and battles.

Key Components of the Server Code

- **Imports and Constants:**
    - The server imports necessary packages such as crypto/sha256, encoding/json, fmt, log, math/rand, net, os, strconv, and strings.
    - Constants are defined for the host, port, connection type, minimum players required for a game, and the path to the Pokémon data file.
- **Data Structures:**
    - User : Represents a user with a username and password.
    - Pokemon, Stats, Profile, Damage, Player, Battle: These structs encapsulate the details of Pokémon, player stats, profiles, and battle mechanics.
- **Main Function:**
    - The server listens for incoming TCP connections on the specified host and port.
    - It accepts client connections and handles authentication.
    - If authentication is successful, it adds the player to a list and waits for enough players to start a game.
- **Authentication:**
    - The authenticate function reads authentication data from the client, verifies it against stored user data, and sends a response back to the client.
- **Game Logic:**
    - Once enough players are connected, the server loads Pokémon data from a JSON file and randomly assigns Pokémon to each player.
    - Players choose their starting Pokémon, and the server initiates a battle between the players.
    - The runBattle function manages the turn-based battle mechanics, allowing players to attack, switch Pokémon, or surrender.
- **Communication:**
    - The server communicates with clients using TCP, sending messages about game state, player actions, and battle results.

Client (Client/Client.go)

The client code allows players to connect to the server and interact with the game. It provides a user interface for players to input commands and view game messages.

Key Components of the Client Code

- **Imports:**

    - The client imports packages such as bufio, fmt, log, net, os, and strings.
- **Constants:**
    - Constants define the host, port, and connection type for the client to connect to the server.
- **Main Function:**
    - The client establishes a connection to the server using the specified host and port.
    - It prompts the user for authentication details (username and password) and sends them to the server.
- **User Input Handling:**
    - The client continuously listens for user commands, allowing players to move, check their inventory, and engage in battles.
    - Commands are sent to the server, and responses are received and displayed to the user.
- **Game Interaction:**
    - The client handles various game interactions, such as:
        - Sending movement commands to the server.
        - Displaying Pokémon encounters and battle results.
        - Allowing players to choose actions during battles

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: Manage TCP connections and player interactions using Go's net package.

Challenge: Authenticating Users

- Solution: Improve the security of the game by implementing a simple authentication system with a username and password.

Challenge: Managing Game State

- Solution: Creating multiple structures for players, pokemons and battles to ensure smooth interactions between players and the server in the game.

Challenge: Ensuring Real-time Communication

- Solution: Managing interactions between clients and servers in real-time using blocking reads and writes on the TCP connection.

4. PokeCat

Introduction

Using UDP for communication, the project entails creating a multiplayer Pokémon game. Multiple users can join a server, navigate the virtual world, find Pokémon, and control their inventories in this game. The server manages Pokémon encounters, player interactions, and game world state maintenance.

Objectives

1. Implement a multiplayer game using UDP for communication.
2. Allow players to move around a 20x20 grid world.
3. Let players capture Pokémon by simulating Pokémon encounters.
4. Use JSON files to store player data and inventory.

Project Structure

Directories

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

Data Structures

- **Stats**: Contains information such as HP, Attack, Defense, etc. of each Pokemon.
- **GenderRatio**: Defines the ratio between male and female pokemons.
- **Profile**: Defines other information like height, weight, catch rate, etc
- **DamageWhenAttacked**: Holds damage multipliers depending on elements when the Pokemon is attacked.

- **Moves**: Contains information about the moves a Pokemon can learn.
- **Pokemon**: Major struct to hold the aforementioned structures.
- **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. Requests a username from the user.
2. Uses UDP to establish a connection to the server.
3. Uses user input to send commands to the server.
4. Listens for and shows the server's answers.

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: To provide thread-safe operations on player data, Go's synx.Mutex was utilized.

Challenge: Managing Game State

● Solution: To monitor the game state, separate maps for players and Pokémon were maintained.

Challenge: UDP Communication

● Solution: To effectively handle UDP communication and manage incoming and outgoing messages, Go's net package was utilized.

Conclusion

The Pokémon projects featured in this study demonstrate Go's adaptability in addressing various programming issues. Each project showcases Go's versatility, from web scraping to real-time multiplayer interactions, showcasing its efficiency, robustness, and appropriateness for many jobs.