VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NET-CENTRIC PROGRAMMING
POKÉMON

By
Truong Duc Duy Khang – ITITIU20227
Ma Phung Nghia – ITCSIU21206

Full Name	Student ID	Contribution
Truong Duc Duy Khang	ITITIU20227	50%
Ma Phung Nghia	ITCSIU21206	50%

TABLE OF CONTENTS

I. INTRODUCTION

The primary objective of this project is to provide an engaging gaming experience that balances performance and creativity, leveraging JSON data handling, terminal-based graphics, and real-time interaction. The report outlines the development process, technical architecture, and implementation details, shedding light on the challenges encountered and solutions devised.

This introduction sets the stage for understanding the game's core functionalities, development environment, and the inspiration behind its creation.

II. POKEDEX

The Pokedex section of the project is responsible for gathering and storing detailed information about Pokémon. The implementation involves web scraping and data extraction from a Pokedex website, specifically using the Playwright library in Go. Below is a detailed explanation of the key components:

1. Overview of the Main Workflow (main.go)

The main entry point for the application initializes the components responsible for crawling Pokémon data (Pokedex.CrawlDriver) and starts the Pokémon catching game (CatTest.StartGame).

```
package main

import (
	CatTest "main/PokeCat"
	"main/Pokedex"
)

func main() {
	Pokedex.CrawlDriver()
	CatTest.StartGame()
	// Get boolean variable "done" from Pokedex
	// If "done" variable is true in Pokedex, call PokeCat.Start()
}
```

2. Pokémon Crawler (Crawler.go)

This module handles scraping data from the Pokedex website. It extracts various details about Pokémon, such as their stats, profile, and evolution details. Here's how the process is structured:

a. CrawlDriver Function

This function:

- Initializes a Playwright instance and launches a Chromium browser.
- Navigates to the Pokedex website (https://pokedex.org/#/).
- Loops through Pokémon entries to extract their details by simulating button clicks.

```
func CrawlDriver() {
    // Initialize Playwright and open the browser
    pw, err := playwright.Run()
    browser, err := pw.Chromium.Launch()
    page, err := browser.NewPage()
    page.Goto(url)

    // Get total Pokémon count and crawl details
    totalPokemons := ExtractPokemonNumber(page)
    fmt.Println("TOTAL POKEMON WILL BE EXTRACTED: ", totalPokemons)

    for i := 0; i < totalPokemons; i++ {
        locator := fmt.Sprintf("button.sprite-%d", i+1)
        button := page.Locator(locator).First()
        if isVisible, _ := button.IsVisible(); isVisible {
            button.Click()
            ExtractPokemon(page)
            page.GoBack()
        }
    }

    // Save results to JSON
    CreateJson(pokemons)
    browser.Close()
```

```
    pw.Stop()
}
```

b. Extracting Pokémon Information

Details such as stats, profile (height, weight, gender ratio, abilities, etc.), and evolution are extracted by navigating the page elements:

- **Extract Stats:** Retrieves stats like HP, Attack, Defense, Speed, and others.

```
func ExtractPokemon(page playwright.Page) {
    stats := Model.Stats{}
    entries, _ := page.Locator("div.detail-stats-row").All()
    for _, entry := range entries {
        title, _ := entry.Locator("span:not([class])").TextContent()
        switch title {
        case "HP":
            stats.HP, _ = strconv.Atoi(entry.Locator("span.stat-bar > div.stat-
bar-fg").TextContent())
        // Additional stats extraction...
        }
    }
    fmt.Println("STATS: ", stats)
}
```

- **Extract Profile:** Collects metadata like height, weight, catch rate, and gender ratio.

```
func ExtractPokemon(page playwright.Page) {
    profile := Model.Profile{}
    entries, _ := page.Locator("div.monster-minutia").All()
    for _, entry := range entries {
        title, _ := entry.Locator("strong:nth-child(1)").TextContent()
        stat, _ := entry.Locator("span:nth-child(2)").TextContent()
        switch title {
        case "Height:":
            profile.Height, _ = strconv.ParseFloat(strings.Split(stat, " ")[0],
32)
        case "Weight:":
```

```
            profile.Weight, _ = strconv.ParseFloat(strings.Split(stat, " ")[0],
32)
        // Additional profile data extraction...
        }
    }
    fmt.Println("PROFILE: ", profile)
}
```

- **Extract Evolution:** Captures information about the Pokémon's evolutionary stages.

```
entries, _ := page.Locator("div.evolutions > div.evolution-row").All()
for _, entry := range entries {
    evolutionLabel, _ := entry.Locator("div.evolution-label >
span").First().TextContent()
    fmt.Println("Evolution: ", evolutionLabel)
}
```

3. Data Storage

Extracted Pokémon data is saved as JSON files using the CreateJson function for further use in the Pokémon catching game.

III. POKECAT

1. Server.go

Purpose:

- Acts as the main entry point for the game.
- Initializes the game world, Pokedex, and spawns Pokémon.
- Manages player login and the transition to game services.

Key Components:

a. Global Variables:

- LoginChannel: Used to synchronize login status.

b. Main Functionality:

- **StartGame():**
  - Initializes the Pokedex and game world.

        - Starts the login service in a separate goroutine.
        - Waits for players to log in via HasLoggedInChannel and initializes them in the game world.

c. Workflow:

- Login credentials are received.
- A new player is created and added to the game.
- Game services are started for the player.

2. Client.go

Purpose:

- Handles the client-side interactions, including logging in, account creation, and communication with the server.

Key Components:

a. Main Functions:

- **StartClient():**
    - Prompts the user to log in or create an account.
    - Establishes a connection with the server.
    - Manages the login flow, including error handling and retries.
- **Authorization():**
    - Sends login or account creation credentials to the server.
    - Handles responses to guide the client through the authentication process.
- **Login() & CreateAccount():**
    - Collect user credentials from the terminal input.
    - Validates input (e.g., password length for account creation).
- **ReceiveMessage():**
    - Continuously listens for server messages and displays them to the user.

b. Workflow:

- Prompts the user for their choice (log in or create account).
- Sends appropriate credentials to the server.
- Handles responses such as success, errors, or retries.

3. LoginService.go

Purpose:

- Handles the backend logic for user login, logout, and account management.
- Manages user persistence using a JSON file.

Key Components:

a. Global Variables:

- activePlayer: Map of active players indexed by their addresses.
- ActivePlayer: Pointer to the currently active player's credentials.
- HasLoggedInChannel: Used to notify the server of successful logins.

b. Main Functions:

- **StartServiceServer():**
  - Sets up a UDP server to listen for client messages.
  - Routes messages to appropriate handlers (e.g., login or logout).
- **handleAuthorization():**
  - Processes login and account creation requests.
  - Validates user credentials against the JSON user database.
  - Updates the database for new account creations.
- **loadUsers() & saveUsers():**
  - Load and save user data to/from a Users.json file.
- **CreateToken():**
  - Generates a random token for session identification.

c. Workflow:

- Server listens for client requests.
- Validates and processes login or account creation.
- Updates activePlayer and notifies the game server of successful logins.

4. Game.go

Key Components

a. Data Structures

- **Coordinate**: Defines X and Y coordinates for positions in the grid.
- **Pokemon**: Represents Pokémon in the game, containing their data and grid position.
- **World**:
    - Tracks the grid (Size).
    - Maintains Pokémon positions in a map[string]*Pokemon (cell map).
    - Tracks players in a map[string]*Player.
    - Uses a mutex (sync.Mutex) to ensure safe concurrent access.
- **Player**: Represents a player with credentials and current position on the grid.

b. Initialization

- **Pokedex**: Loaded from a JSON file (POKEMONS.json).
- **World**: Configured to a default grid size (20x20) with empty cells and no players initially.
- **Player**: Spawns at a random position on the grid.

c. Gameplay Features

- **Spawning Pokémon**:
    - A fixed number of Pokémon (ReleasePokemon) are placed randomly on the grid.
    - Uses a map to ensure unique cell identifiers (key = fmt.Sprintf("%d,%d", x, y)).
- **Capturing Pokémon**:
    - If a player moves to a cell with a Pokémon, it's captured, removed from the grid, and added to the player's collection.
    - Captured Pokémon are written back to Users.json.

d. Multiplayer Interaction

- Supports multiple players:
    - **Player 1 Controls**: Arrow keys.
    - **Player 2 Controls**: WASD keys.

    - Thread-safe updates using sync.Mutex.
- **Player Movement**:
    - Players navigate the grid while avoiding going out of bounds.
    - Each player's movement triggers a check for Pokémon capture.

e. Rendering

- Uses the termbox-go library for terminal-based rendering:
    - Pokémon: Rendered as red 'O'.
    - Players: Rendered as green or custom-colored 'P'.
    - Empty spaces: Rendered as white '.'.
- Smooth rendering achieved by reducing flicker with frame buffering and locking during rendering.

IV. CONCLUSION

This project successfully developed a 2D Pokémon-inspired multiplayer game, blending engaging gameplay with technical efficiency. By leveraging the Go programming language, terminal-based rendering, and robust concurrency mechanisms, the game achieved real-time interaction, dynamic Pokémon spawning, and seamless player movement.

The project highlights the importance of simplicity and performance in game development, showcasing an accessible yet interactive gaming experience. Despite some challenges in user interface design and synchronization, the solutions implemented demonstrate a strong foundation for future enhancements, such as expanding features, improving graphics, and enabling web-based deployment.