University of Perpetual Help System Laguna
City of Biñan, Laguna, Philippines, 4024
(02)779-5310 • (049)554-5150 • (049)554-5161
www. uphsl.edu.ph

UPHSL College of Computer Studies

Binan, Laguna

Who’s That Pokemon?

Programming Languages Laboratory

Bachelor of Science in Computer Science

J4A

Submitted to

Prof. Grachel Eliza Greta

Submitted by

Kelvin Adam Aninang
Angela Cabanes
Michaela Jornaes

November 2025

Executive Summary

The "Who's That Pokémon?" application is a desktop-based interactive trivia system designed to test and challenge a user's knowledge of the original Pokémon characters through visual silhouette identification. Its primary purpose is to provide an engaging, gamified environment that transcends static trivia formats, replacing them with a dynamic gameplay experience that integrates real-time data fetching and robust user experience elements.

The system's core features are built around a flexible and configurable quiz flow. The experience begins with a refined Start Menu that clearly separates the two primary modes: Game Start and Catalogue Viewer. Selecting "Start Game" leads to a multi-tiered Difficulty System (Easy, Medium, Hard) that dynamically adjusts the countdown timers, demonstrating system flexibility. During gameplay, the system includes essential game mechanics such as lives tracking, an in-game scoring system, and a hint mechanism. Crucially, the UI employs smooth, non-disruptive feedback for guesses and timeouts, displaying results directly on the screen for a few seconds before seamlessly advancing to the next round.

A major feature enhancing the application's usability and flexibility is the Catalogue Viewer. This standalone screen allows users to review the complete list of Pokémon available in the quiz (Generation 1). This catalogue concurrently loads data, displaying each Pokémon's ID, sprite image, and type information within a fixed, scrollable window, addressing the need for pre-game review or reference. This concurrent data fetching and structured UI layout demonstrates advanced software engineering practices, specifically handling heavy I/O operations without freezing the graphical interface.

The application is built using Python, leveraging the Tkinter library for a modular graphical user interface (GUI) and the Pillow (PIL) library for dynamic image manipulation (creating silhouettes). The use of Python's threading module for all API data fetching and processing—including generating the silhouette images for the quiz and fetching all data for the catalogue—shows effective use of concurrency and optimization to prevent UI hangs. The primary target users are nostalgia-seeking Pokémon enthusiasts and casual gamers, but the modular, documented, and concurrently optimized code also makes it a valuable case study for computer science students analyzing effective object-oriented design and performance management in Python.

System Overview

Background

The project originated from the enduring popularity of the "Who's That Pokémon?" segment featured in the original Pokémon animated series. This segment involved presenting the silhouette of a Pokémon and challenging viewers to identify it. The idea was to transform this simple, nostalgic concept into a modern, interactive desktop application. The system serves as a playful test of visual memory and trivia knowledge for fans, while simultaneously acting as a platform to demonstrate the mastery of Python programming concepts and contemporary software engineering practices, particularly in the areas of concurrency and UI/UX design.

Problem Addressed

The system addresses the following problems:

- **Static Trivia Engagement:** Traditional, text-based trivia games often lack dynamic engagement and visual appeal, leading to user fatigue. The system replaces static questions with a visual puzzle and a time-constrained, gamified format.
- **Lack of Concurrency in UI:** Developing responsive desktop applications often faces the challenge of UI freezing when performing time-consuming tasks (like fetching data from external APIs or complex image manipulation). The system solves this by implementing Python threading to handle all I/O and processing tasks asynchronously.
- **Unstructured Learning/Review:** Users interested in Pokémon trivia or game data often rely on static websites. The inclusion of the Catalogue Viewer provides a structured, integrated, and concurrently loaded feature for pre-game review and post-game reference.

Objectives of the Developed Solution

The primary objectives achieved by the developed solution are:

1. **Integrate Core Programming Concepts:** Successfully merge Object-Oriented Programming (OOP), modularity (separate files for UI, logic, and API), and **c**oncurrency (threading) into a single, cohesive, and maintainable application.
2. **Enhance User Experience (UX):** Implement improved flow control through a clear Start Menu and a smooth, non-disruptive feedback system for correct and incorrect answers, replacing traditional alert boxes.
3. **Expand Functionality and Flexibility:** Introduce a comprehensive new feature, the Catalogue Viewer, which concurrently loads and displays Pokémon sprites and type data, thereby expanding the system's utility beyond just the quiz.

4. **Demonstrate Performance Optimization:** Ensure the Graphical User Interface (GUI) remains responsive by performing heavy tasks, such as API calls and image processing (silhouette creation), in dedicated background threads.

Development Environment and Tools

Category	**Tool / Version**	**Purpose in Project**
Programming Language	Python 3.11+	Core language for application logic and backend processes.
Graphical User Interface	Tkinter (Standard Library)	Used for building the desktop GUI (windows, buttons, labels).
Image Processing	Pillow (PIL)	Used for runtime image manipulation, specifically creating the **silhouette effect** for the quiz and resizing sprites for the catalogue.
Data Fetching	requests library	Handles asynchronous communication with the external PokéAPI to retrieve Pokémon data and images.
Concurrency	threading (Standard Library)	Ensures the UI remains responsive by running all API and image tasks on **background threads**.
Code Editor & IDE	Visual Studio Code (VS Code)	Primary environment for coding, debugging, and file management.
Version Control	Git and GitHub Classroom	Manages source code version history, collaboration, and submission of deliverables.

Software Design & Architecture

Diagram of Program Structure

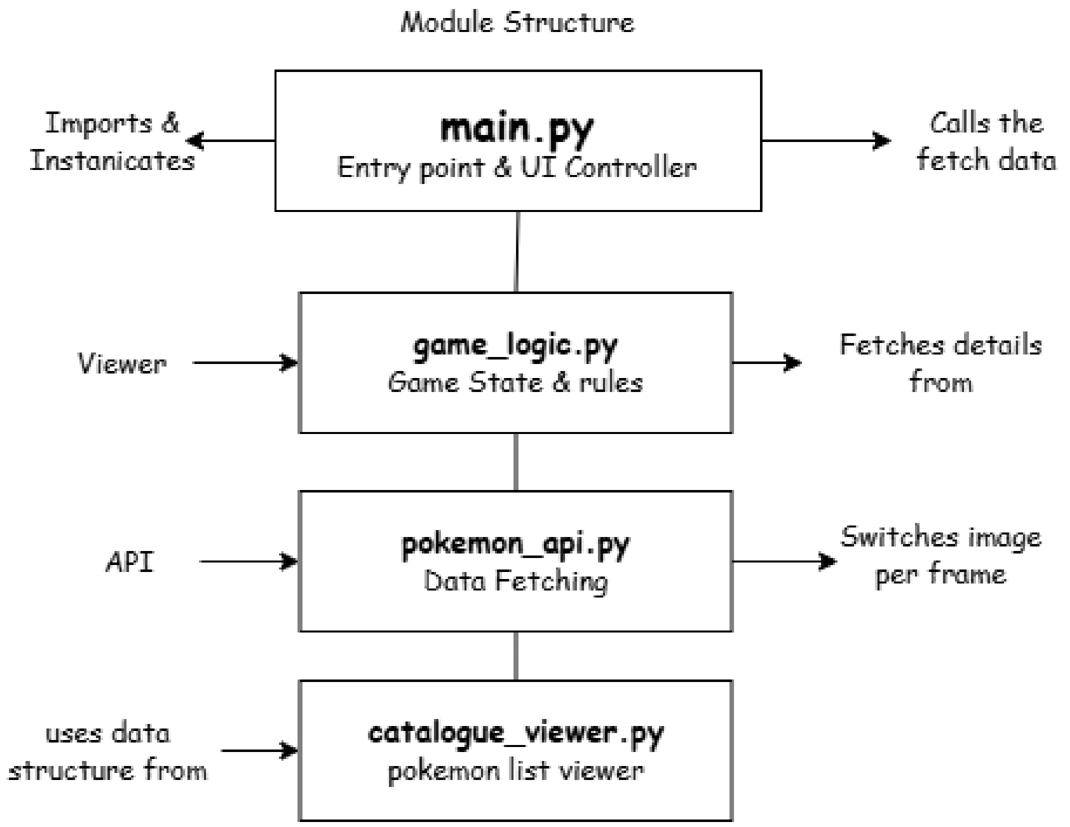


Figure 1. Model Structure

The module structure of the "Who's That Pokémon?" program consists of four Python files. main.py serves as the entry point and user interface (UI) controller, where it handles screen switching, timer management, and instantiates the other classes. The game_logic.py module manages the core game state and rules, tracking lives, score, and checking user guesses, with main.py fetching these details for display. The pokemon_api.py module is responsible for API data fetching and concurrent image processing, specifically fetching a random Pokémon's data and generating its silhouette for the game. Finally, the catalogue_viewer.py module handles the concurrent fetching of the complete Pokémon list and is used by the main.py UI controller to display the full, scrollable catalogue view.

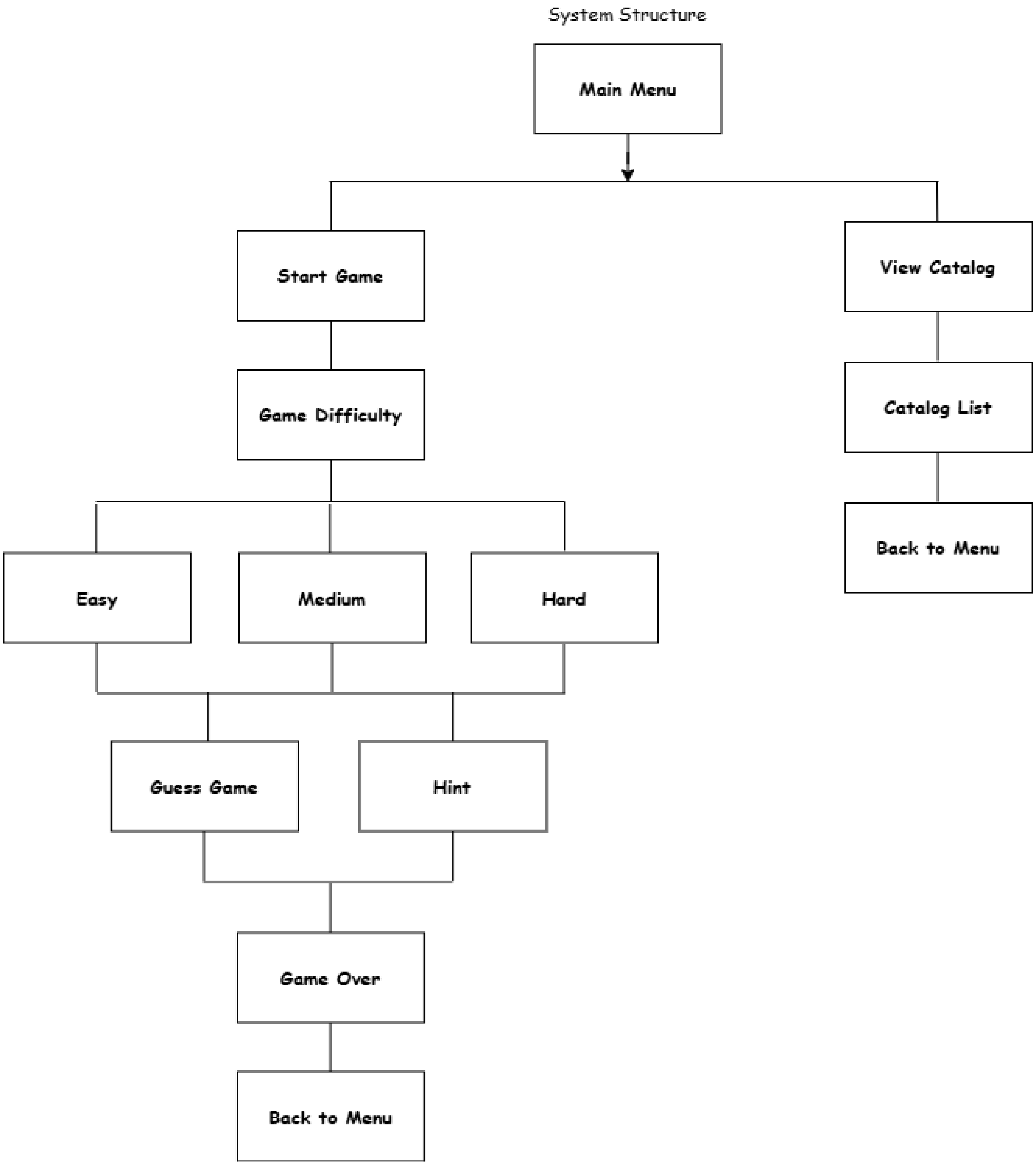


Figure 2. System Structure

The system structure of the application is organized around two main user paths originating from the Main Menu.

The primary path is the Game Flow, which starts when the user selects Start Game. This leads to the Game Difficulty selection, offering three modes: Easy, Medium, and Hard. Regardless of the chosen difficulty, the user enters the core gameplay loop, which includes the Guess Game feature and the optional Hint feature. The game concludes with the Game Over screen, from which the user can return to the Main Menu.

The secondary path is the Catalogue Flow, accessed by selecting View Catalog from the Main Menu. This immediately displays the Catalog List. From the catalogue, the user can navigate back to the Main Menu.

Explanation of Major Functions/Subprograms

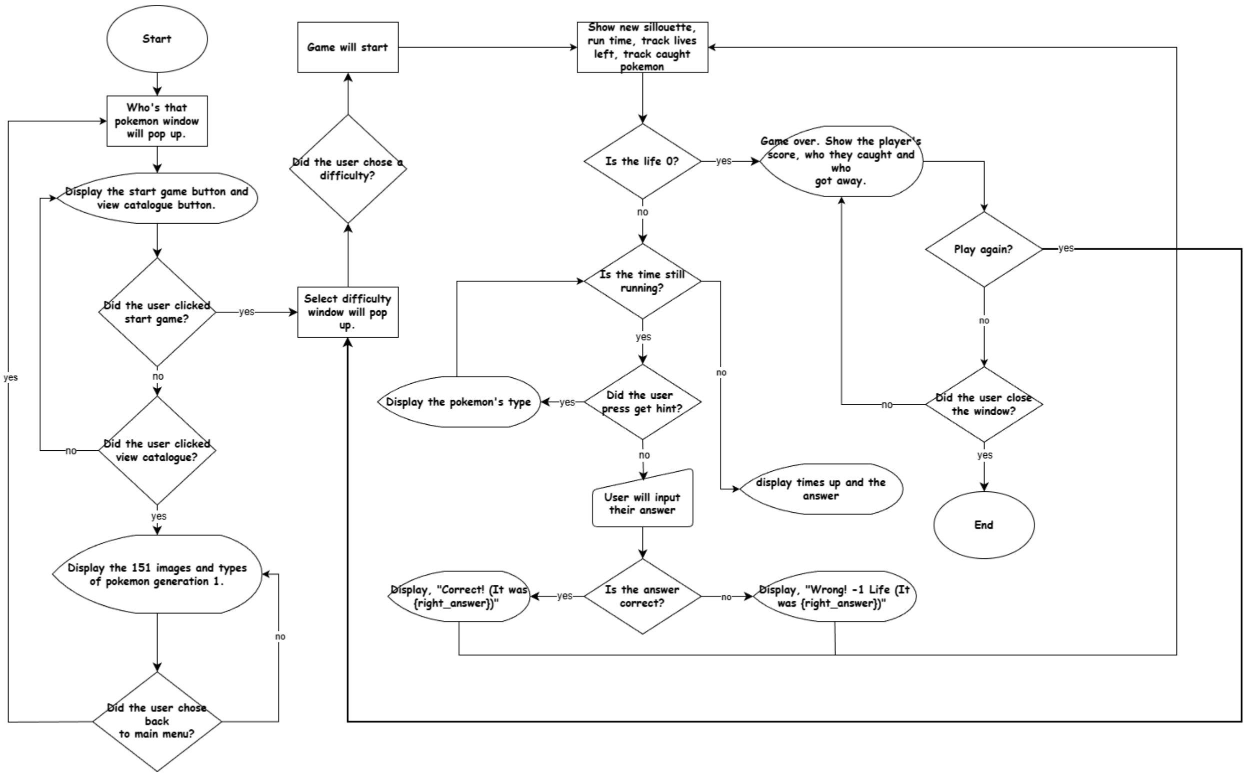
The improvements to the "Who's That Pokémon?" application are based on three main ideas: Concurrency, Modularity, and Abstraction.

Concurrency means the app can do many things at once. We used this to solve the problem where the screen would freeze while the app waited for the internet to send Pokémon data or while the app created the silhouette images. We used Python's Threading tool to send these slow jobs—like getting data from the PokéAPI or loading 151 catalogue sprites—to a separate Worker Thread. This makes sure the main screen stays smooth and responsive. For example, if you click the "View Catalogue" button, the loading happens in the background, so the app doesn't crash or freeze while waiting. This gives the player a professional and stable experience.

Modularity means we organized the code into small, specialized files. Instead of putting everything in one long file, we separated the work: main.py only handles the screen flow (like moving from the Start Menu to the Game Screen); game_logic.py only handles the rules (like counting lives and calculating the score); and other modules like pokemon_api.py and catalogue_viewer.py only handle getting and preparing data from the internet. This separation makes the app easier to fix and update in the future because if you change a game rule, you only touch one small file (game_logic.py).

Abstraction means we hide the complicated details. For example, when the game needs a new Pokémon silhouette, the main screen simply calls a function named get_random_pokemon(). The main screen doesn't need to know the messy steps involved: it doesn't care about connecting to the API, dealing with network errors, downloading the image, or turning the image black. All those complex steps are hidden inside the function. This makes the code very clean, safe, and easy to use, since each part only sees the necessary information, not the messy parts. These improvements ensure the final system is well-built, fast, and easy to maintain.

Flowchart or Pseudocode of Main Processes


Start
Who's that pokemon window will pop up.
Display the start game button and view catalogue button.
Did the user clicked start game?
yes
no
Did the user clicked view catalogue?
no
yes
Display the 151 images and types of pokemon generation 1.
Did the user chose back to main menu?
no
yes
Select difficulty window will pop up.
Did the user chose a difficulty?
Game will start
Show new sillouette, run time, track lives left, track caught pokemon
Is the life 0?
yes
no
Game over. Show the player's score, who they caught and who got away.
Is the time still running?
yes
no
Did the user press get hint?
yes
Display the pokemon's type
no
User will input their answer
display times up and the answer
Is the answer correct?
yes
Display, "Correct! (It was {right_answer})"
no
Display, "Wrong! -1 Life (It was {right_answer})"
Play again?
yes
no
Did the user close the window?
no
yes
End

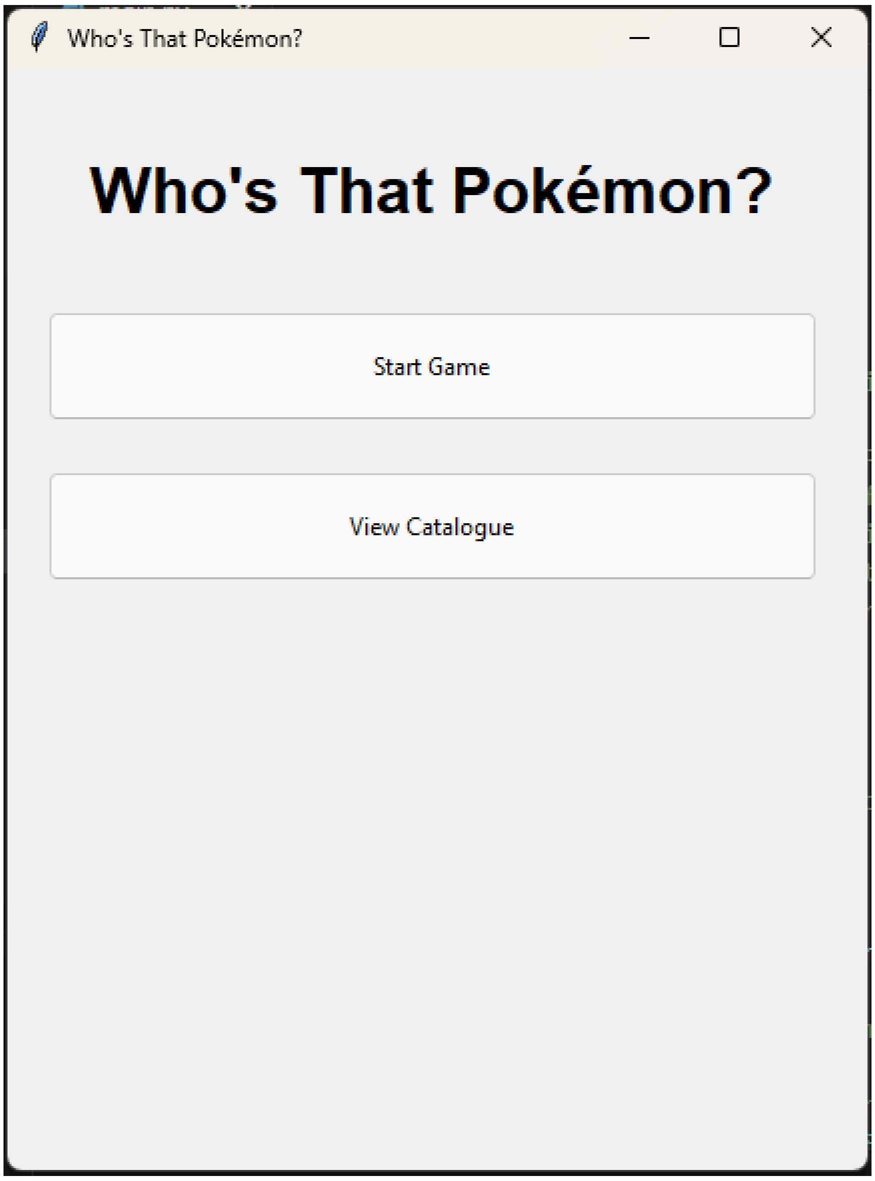
Link: https://drive.google.com/file/d/1N45e3gmNlGKYZ1ikiWk_Ickut5O0MyfH/view?usp=sharing

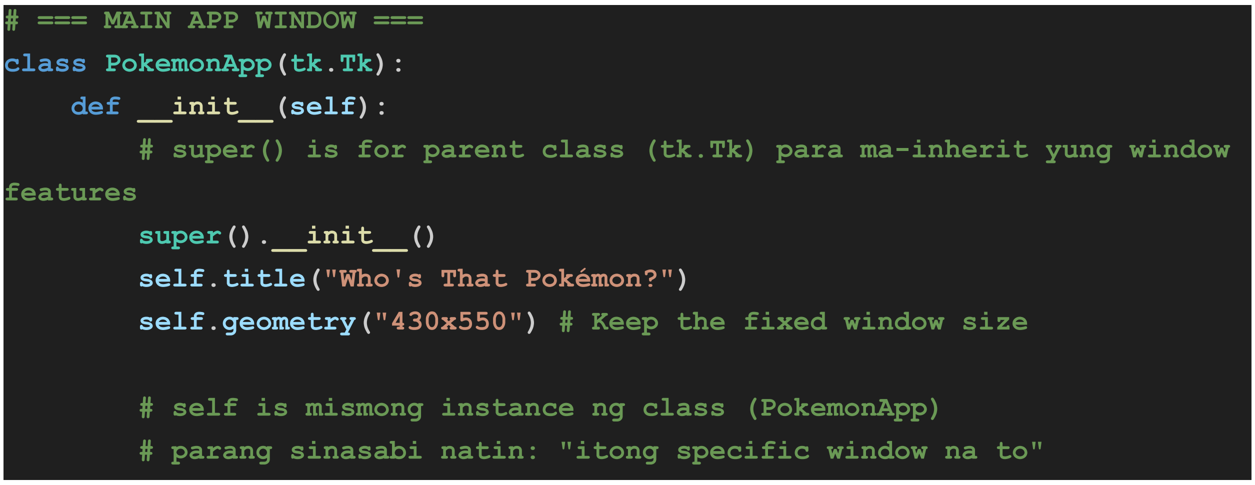
Concurrency or Multi-threading Model

The "Who's That Pokémon?" application uses Python's threading library as its concurrent model. The main purpose is to keep the application fast and responsive by preventing the screen from freezing. This is achieved by separating all slow jobs from the main application thread. Whenever the system needs to perform a blocking task, such as sending a request over the internet to the PokéAPI (network I/O) or doing heavy image processing to create the silhouette, it immediately sends that job to a separate Worker Thread. This worker thread handles the waiting and the CPU processing for both the single quiz Pokémon and the entire 151-entry Catalogue. Because the slow work happens in the background, the main screen (the UI, timer, and buttons) remains fully active. When the worker thread is finished, it uses a special, safe message (a callback mechanism) to tell the main screen to update the image and restart the timer. This design ensures a professional, non-freezing user experience.

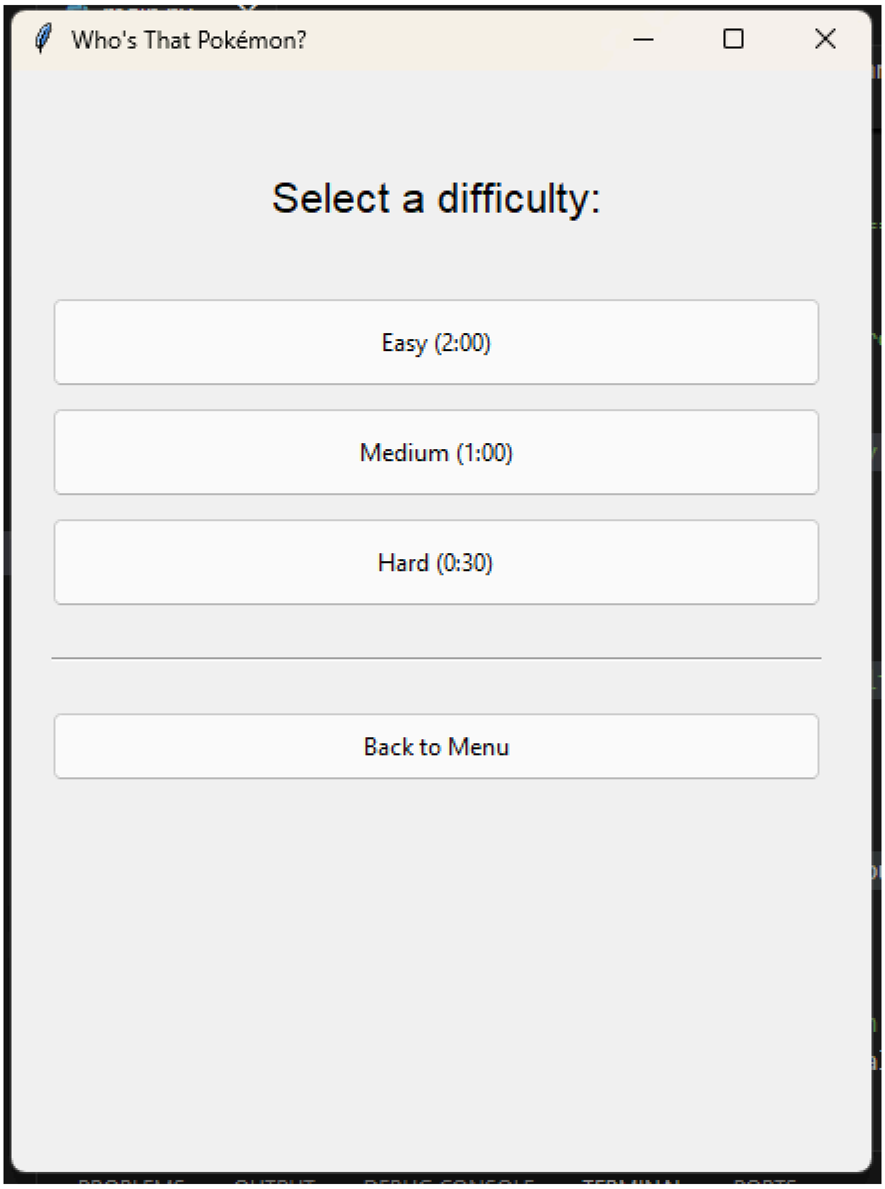
Implementation

Screenshots below represent the ample input/output screenshots or terminal logs, and highlight sections of code that demonstrate: control flow logic, function modularity, and concurrency.



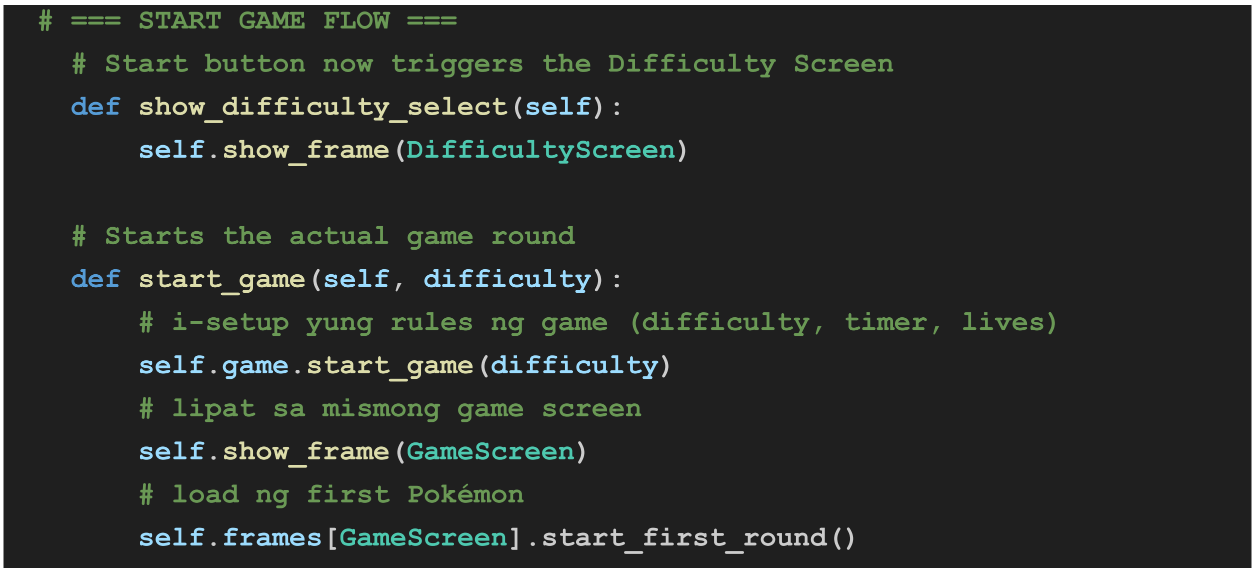


```
# === MAIN APP WINDOW ===
class PokemonApp(tk.Tk):
    def __init__(self):
        # super() is for parent class (tk.Tk) para ma-inherit yung window
features
        super().__init__()
        self.title("Who's That Pokémon?")
        self.geometry("430x550") # Keep the fixed window size

        # self is mismong instance ng class (PokemonApp)
        # parang sinasabi natin: "itong specific window na to"
        self.game = PokemonGame()  # game logic instance galing sa ibang module
        self.catalogue_data = None # Store catalogue data once fetched
        self.catalogue_photo_refs = [] # to prevent garbage collection of images

        # Timer variables para sa sabay-sabay na processes (concurrency)
        self.timer_id = None
        self.remaining_time = 0 # coundown
        self.pokemon_silhouette_tk = None #store the Tkinter image object kasi it will disappear if not

        # Container na mag-hohold ng multiple frames (screens)
        self.container = ttk.Frame(self)
        self.container.pack(fill="both", expand=True)

        # Dictionary ng screens (para madali magpalit)
        self.frames = {}
        # Changed the first screen to StartMenuScreen
        for F in (StartMenuScreen, DifficultyScreen, GameScreen, GameOverScreen, CatalogueScreen):
            # Gumagawa ng instance ng bawat screen
            frame = F(self.container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")
        # this code just places the screen on top of each other. parang overlap lang.

        # Unang screen na lalabas
        self.show_frame(StartMenuScreen)
```

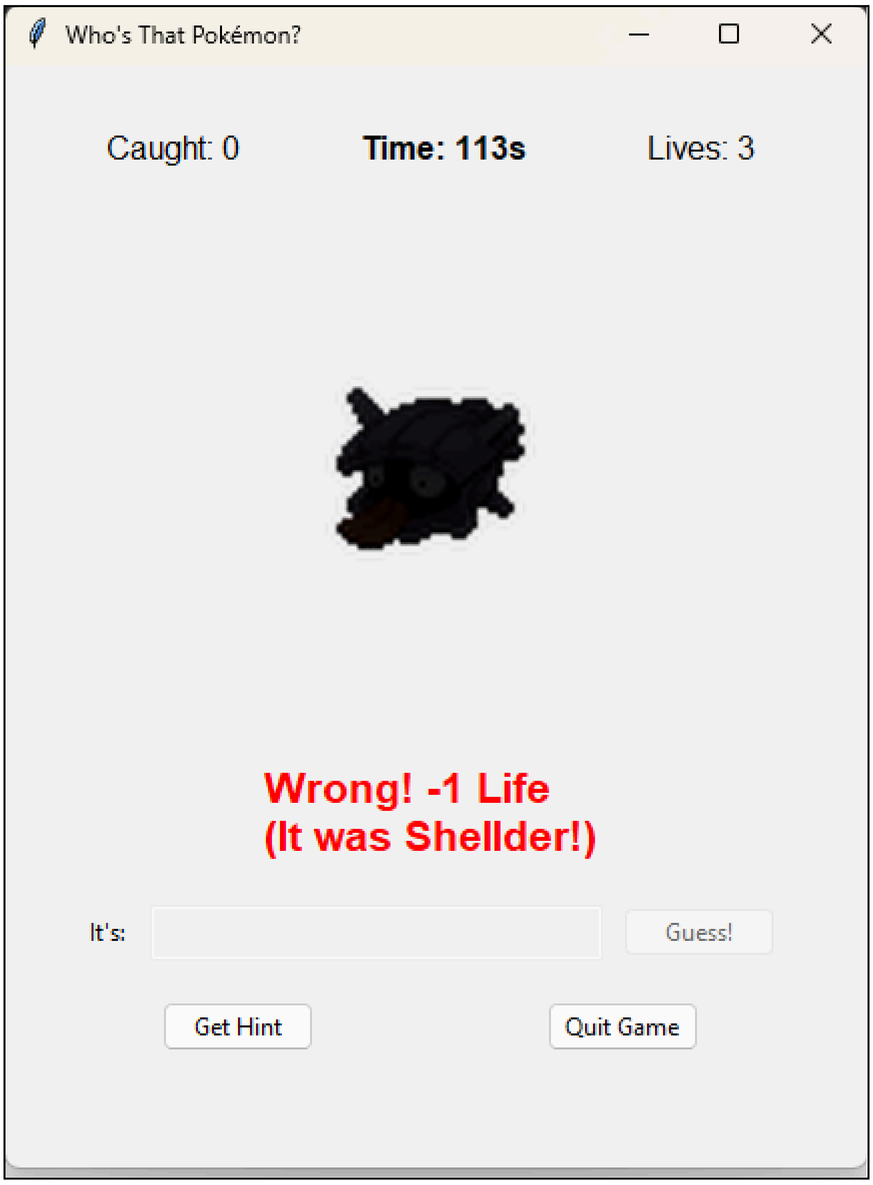


Who's That Pokémon?

Select a difficulty:

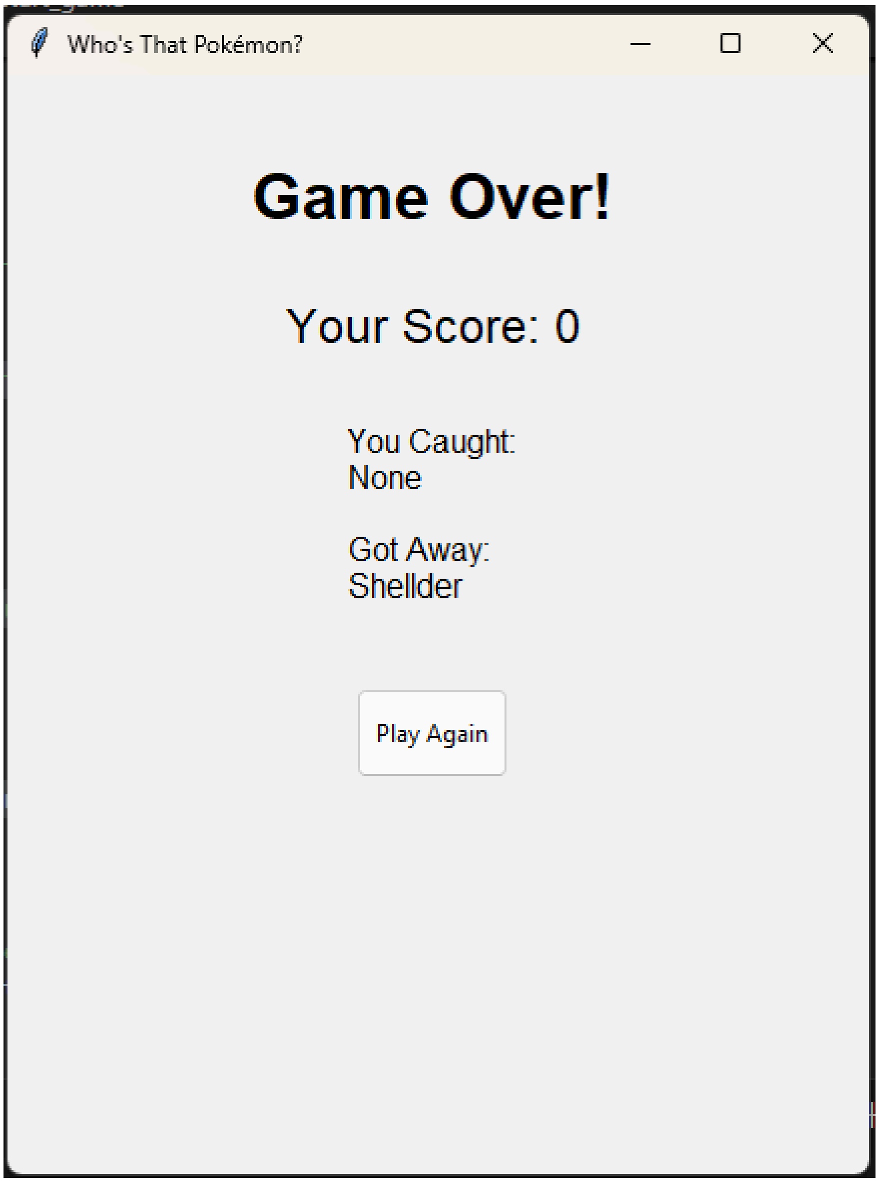


```
# === START GAME FLOW ===
  # Start button now triggers the Difficulty Screen
  def show_difficulty_select(self):
      self.show_frame(DifficultyScreen)

  # Starts the actual game round
  def start_game(self, difficulty):
      # i-setup yung rules ng game (difficulty, timer, lives)
      self.game.start_game(difficulty)
      # lipat sa mismong game screen
      self.show_frame(GameScreen)
      # load ng first Pokémon
      self.frames[GameScreen].start_first_round()
```



Wrong! -1 Life
(It was Shellder!)

```
def start_first_round(self):
        # reset score/lives display sa simula ng game
        self.lives_label.config(text=f"Lives: 
{self.controller.game.lives}")
        self.score_label.config(text=f"Caught: 0")
        self.load_next_pokemon()

    def display_result(self, message, color, full_reveal_message=None, 
reveal=False):
        """
        Displays the result message on the screen for 2 seconds and 
handles the flow.
        """
        self.controller.stop_timer()

        # eto yung part na nagsshow kung ano yung tamang sagot
        # Update UI with the result
        self.result_label.config(text=message, foreground=color)
        self.guess_entry.config(state="disabled")
        self.guess_button.config(state="disabled")

        if reveal and full_reveal_message:
            # Display the full name if it was a timeout/wrong guess that
ends the round

self.result_label.config(text=f"{message}\n({full_reveal_message})")
```



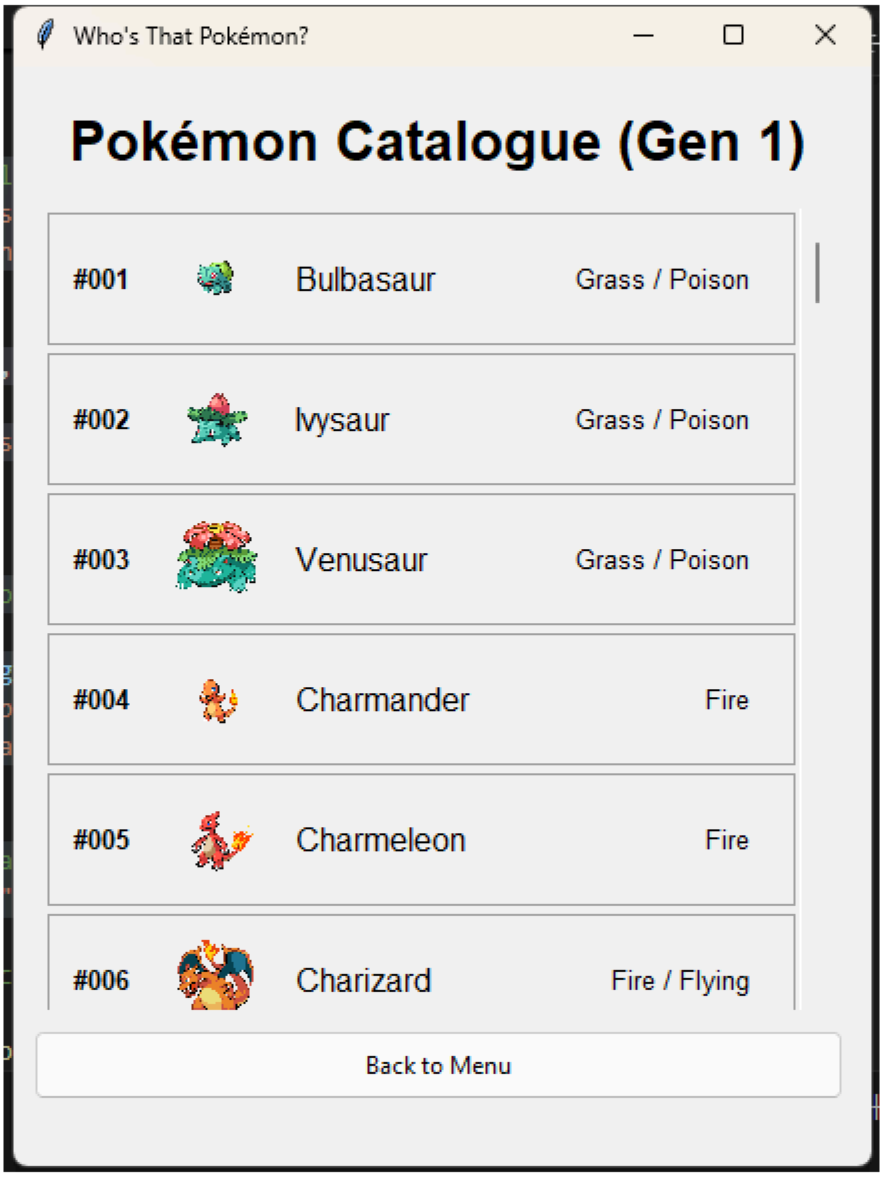
Game Over!

Your Score: 0

You Caught:
None

Got Away:
Shellder

```
  # === END GAME ===
    def end_game(self):
        self.stop_timer()
        self.game.game_active = False
        # display results sa Game Over screen
        self.frames[GameOverScreen].display_results()
        self.show_frame(GameOverScreen)
```

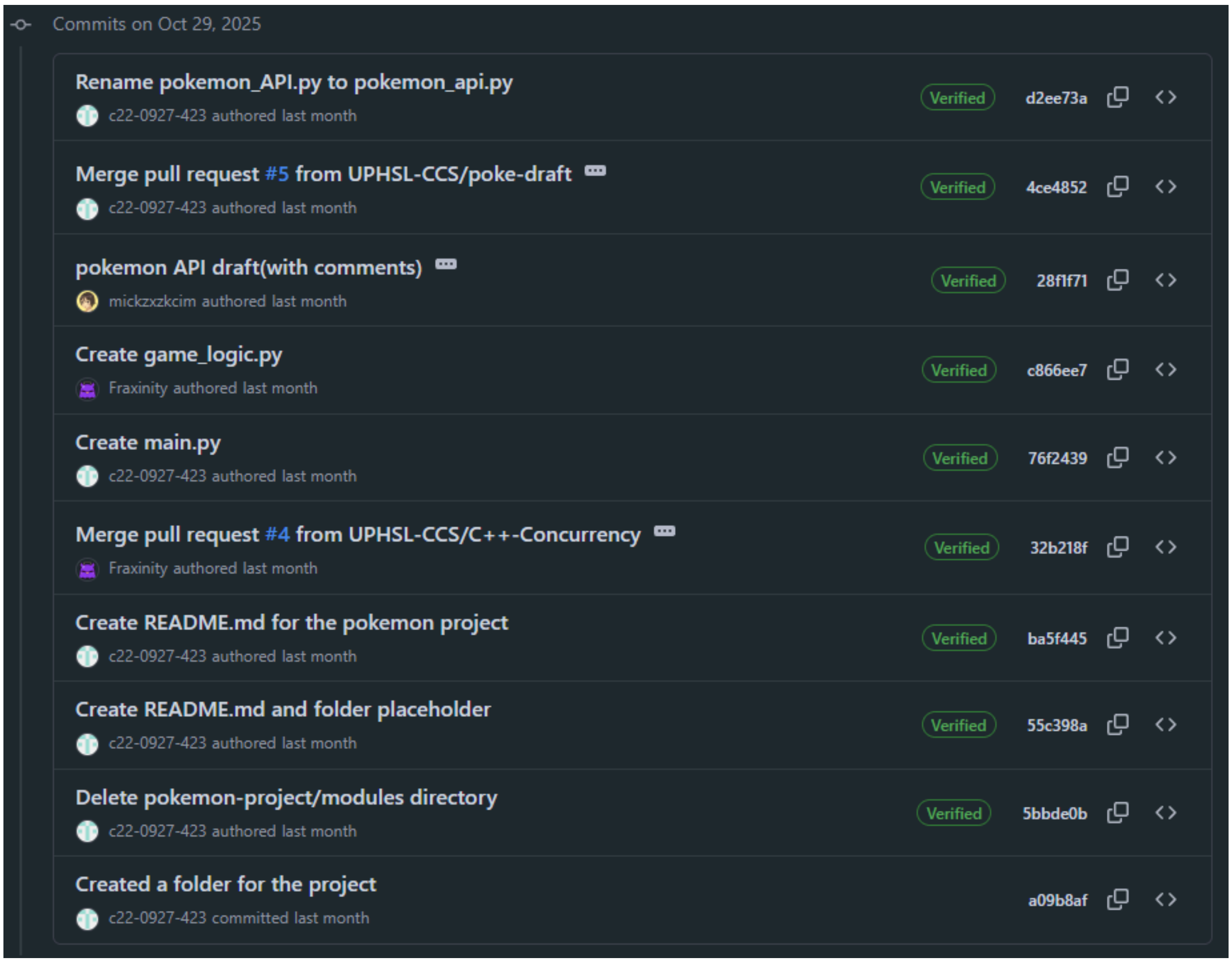


Pokémon Catalogue (Gen 1)

#001	Bulbasaur	Grass / Poison
#002	Ivysaur	Grass / Poison
#003	Venusaur	Grass / Poison
#004	Charmander	Fire
#005	Charmeleon	Fire
#006	Charizard	Fire / Flying

```
class CatalogueScreen(ttk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, padding=10)
        self.controller = controller

        # Title
        ttk.Label(self, text="Pokémon Catalogue (Gen 1)", font=("Arial", 
20, "bold")).pack(pady=10)

        # --- Setup Scrollable Canvas ---
        scroll_container = ttk.Frame(self)
        scroll_container.pack(fill="both", expand=True, padx=5, pady=5)

        # Create Canvas and Scrollbar
        self.canvas = tk.Canvas(scroll_container, borderwidth=0,
highlightthickness=0, height=400)
        self.scrollbar = ttk.Scrollbar(scroll_container,
orient="vertical", command=self.canvas.yview)

        # Inner frame to hold the catalogue items
        self.scrollable_frame = ttk.Frame(self.canvas)

        # 1. TELL THE CANVAS TO SCROLL THE FRAME
        self.scrollable_frame.bind(
            "<Configure>",
            lambda e:
self.canvas.configure(scrollregion=self.canvas.bbox("all"))
        )

        #Create the window inside the canvas

        self.window_id = self.canvas.create_window((0, 0),
window=self.scrollable_frame, anchor="nw")

        # Force the inner frame to match the Canvas width
        # This ensures the list items shrink slightly to make room for the
scrollbar
        self.canvas.bind(
            '<Configure>',
            lambda e: self.canvas.itemconfig(self.window_id,
width=e.width)
        )

        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        #Scrollbar on right, Canvas fills the rest
        self.scrollbar.pack(side="right", fill="y")
        self.canvas.pack(side="left", fill="both", expand=True)

        # Back Button
        ttk.Button(self, text="Back to Menu", command=lambda:
controller.show_frame(StartMenuScreen)).pack(fill="x", ipady=5, pady=(5,
10))

    def load_catalogue(self, data):
        # Clear existing widgets
        for widget in self.scrollable_frame.winfo_children():
            widget.destroy()
        self.controller.catalogue_photo_refs = []
```

Git Commits



Commits on Oct 29, 2025

Rename pokemon_API.py to pokemon_api.py
c22-0927-423 authored last month
Verified d2ee73a

Merge pull request #5 from UPHSL-CCS/poke-draft
c22-0927-423 authored last month
Verified 4ce4852

pokemon API draft(with comments)
mickzxzkcim authored last month
Verified 28f1f71

Create game_logic.py
Fraxinity authored last month
Verified c866ee7

Create main.py
c22-0927-423 authored last month
Verified 76f2439

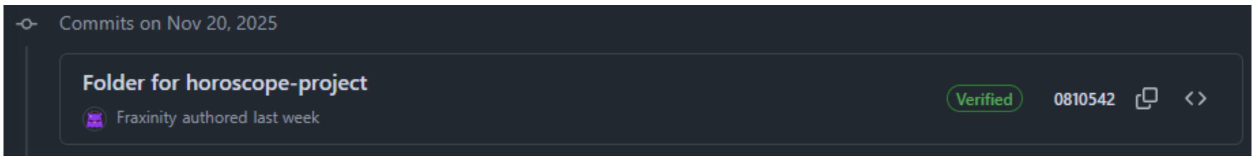
Merge pull request #4 from UPHSL-CCS/C++-Concurrency
Fraxinity authored last month
Verified 32b218f

Create README.md for the pokemon project
c22-0927-423 authored last month
Verified ba5f445

Create README.md and folder placeholder
c22-0927-423 authored last month
Verified 55c398a

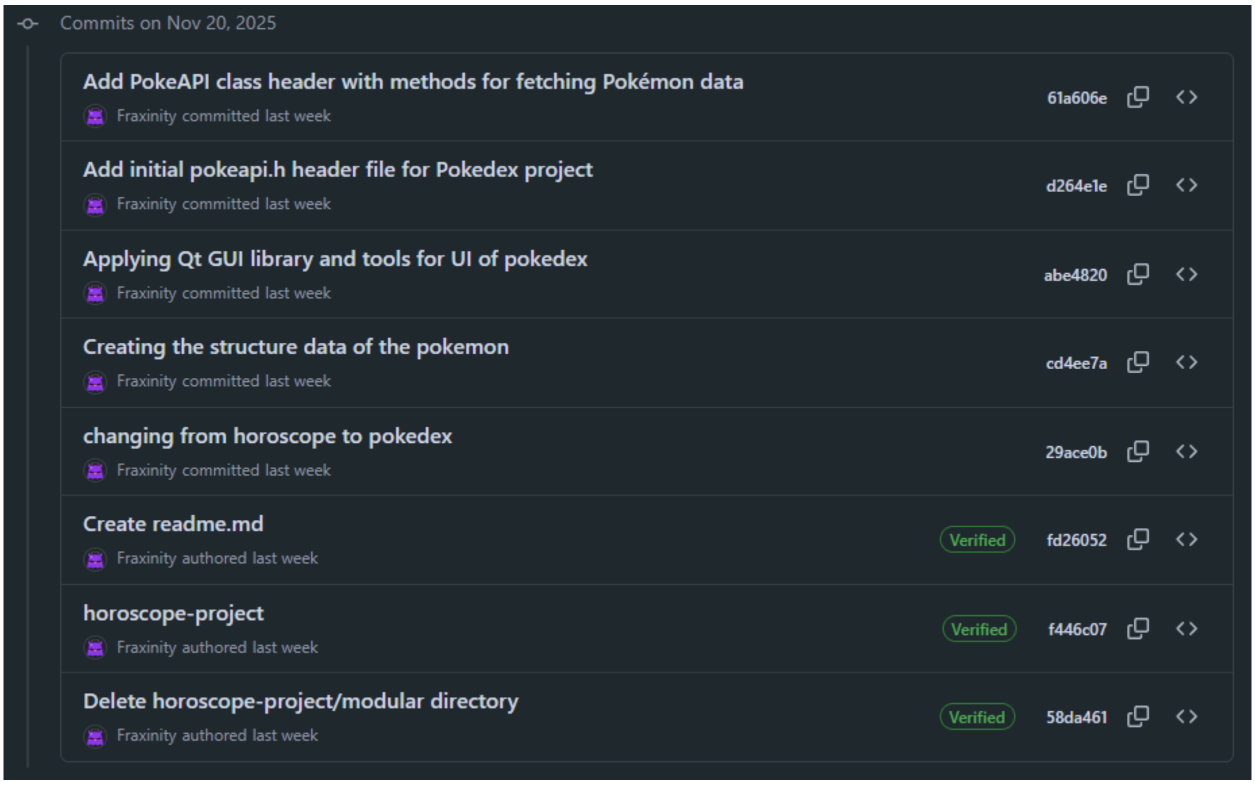
Delete pokemon-project/modules directory
c22-0927-423 authored last month
Verified 5bbde0b

Created a folder for the project
c22-0927-423 committed last month
a09b8af



Commits on Nov 20, 2025

Folder for horoscope-project
Fraxinity authored last week
Verified 0810542



Commits on Nov 20, 2025

Add PokeAPI class header with methods for fetching Pokémon data
Fraxinity committed last week — 61a606e

Add initial pokeapi.h header file for Pokedex project
Fraxinity committed last week — d264e1e

Applying Qt GUI library and tools for UI of pokedex
Fraxinity committed last week — abe4820

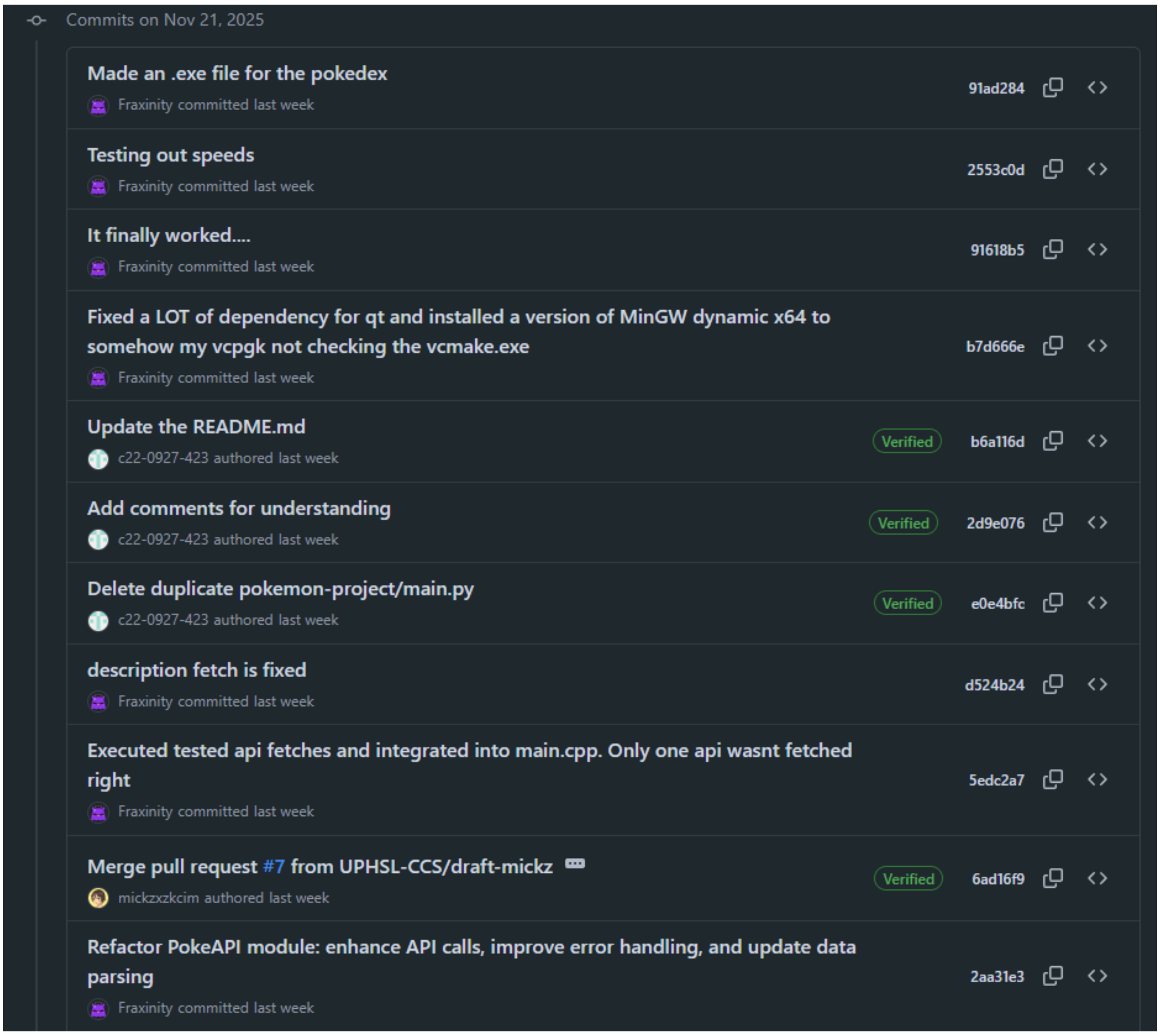
Creating the structure data of the pokemon
Fraxinity committed last week — cd4ee7a

changing from horoscope to pokedex
Fraxinity committed last week — 29ace0b

Create readme.md
Fraxinity authored last week — Verified — fd26052

horoscope-project
Fraxinity authored last week — Verified — f446c07

Delete horoscope-project/modular directory
Fraxinity authored last week — Verified — 58da461



Commits on Nov 21, 2025

Made an .exe file for the pokedex
Fraxinity committed last week — 91ad284

Testing out speeds
Fraxinity committed last week — 2553c0d

It finally worked....
Fraxinity committed last week — 91618b5

Fixed a LOT of dependency for qt and installed a version of MinGW dynamic x64 to somehow my vcpgk not checking the vcmake.exe
Fraxinity committed last week — b7d666e

Update the README.md
c22-0927-423 authored last week — Verified — b6a116d

Add comments for understanding
c22-0927-423 authored last week — Verified — 2d9e076

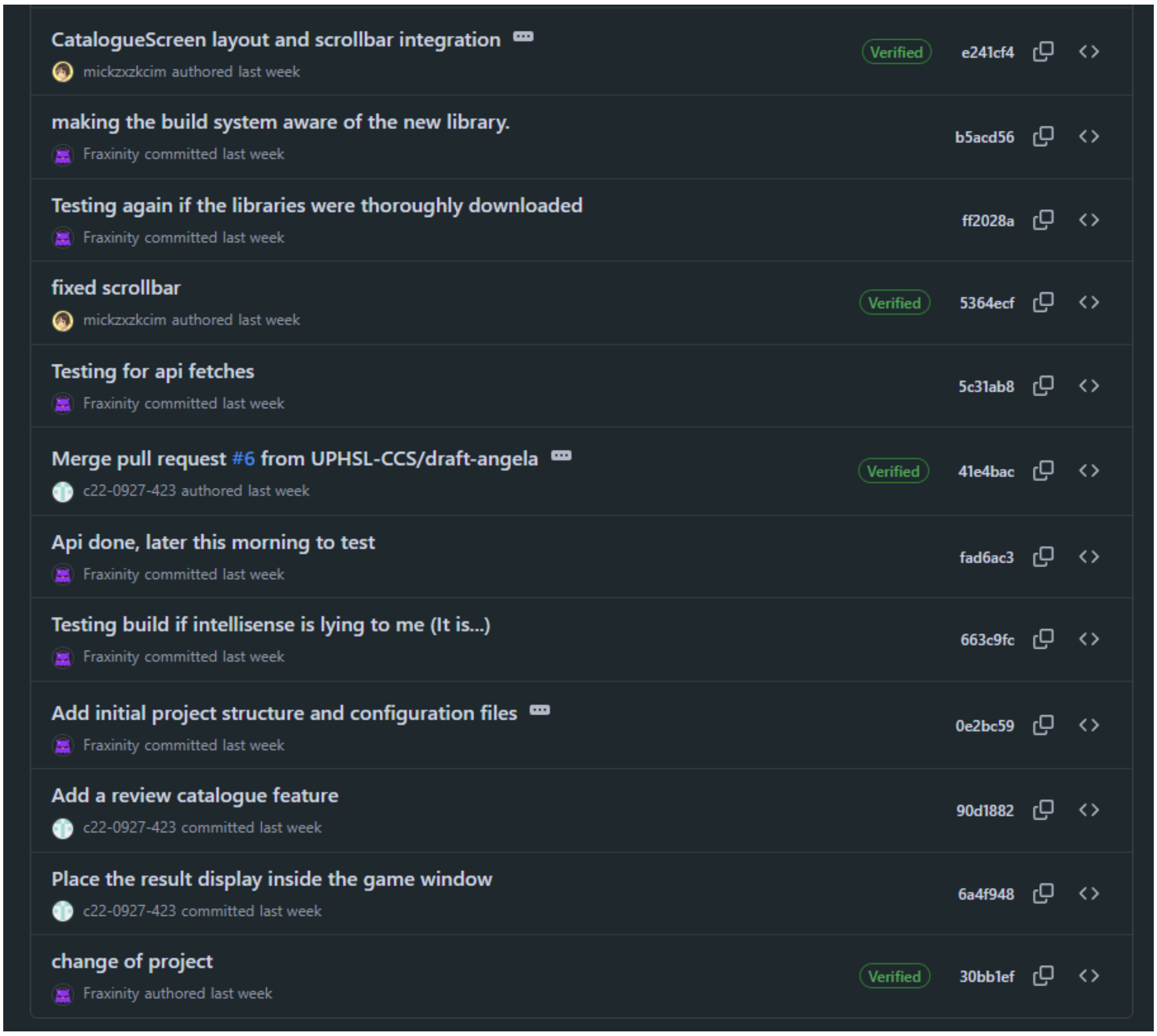
Delete duplicate pokemon-project/main.py
c22-0927-423 authored last week — Verified — e0e4bfc

description fetch is fixed
Fraxinity committed last week — d524b24

Executed tested api fetches and integrated into main.cpp. Only one api wasnt fetched right
Fraxinity committed last week — 5edc2a7

Merge pull request #7 from UPHSL-CCS/draft-mickz
mickzxzkcim authored last week — Verified — 6ad16f9

Refactor PokeAPI module: enhance API calls, improve error handling, and update data parsing
Fraxinity committed last week — 2aa31e3



CatalogueScreen layout and scrollbar integration
mickzxzkcim authored last week — Verified — e241cf4

making the build system aware of the new library.
Fraxinity committed last week — b5acd56

Testing again if the libraries were thoroughly downloaded
Fraxinity committed last week — ff2028a

fixed scrollbar
mickzxzkcim authored last week — Verified — 5364ecf

Testing for api fetches
Fraxinity committed last week — 5c31ab8

Merge pull request #6 from UPHSL-CCS/draft-angela
c22-0927-423 authored last week — Verified — 41e4bac

Api done, later this morning to test
Fraxinity committed last week — fad6ac3

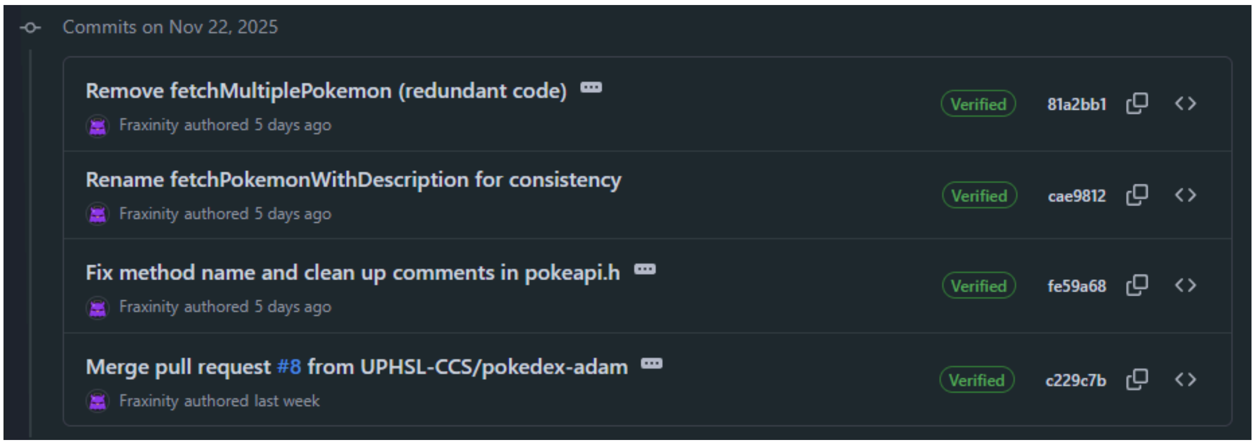
Testing build if intellisense is lying to me (It is...)
Fraxinity committed last week — 663c9fc

Add initial project structure and configuration files
Fraxinity committed last week — 0e2bc59

Add a review catalogue feature
c22-0927-423 committed last week — 90d1882

Place the result display inside the game window
c22-0927-423 committed last week — 6a4f948

change of project
Fraxinity authored last week — Verified — 30bb1ef



Commits on Nov 22, 2025

Remove fetchMultiplePokemon (redundant code)
Fraxinity authored 5 days ago — Verified — 81a2bb1

Rename fetchPokemonWithDescription for consistency
Fraxinity authored 5 days ago — Verified — cae9812

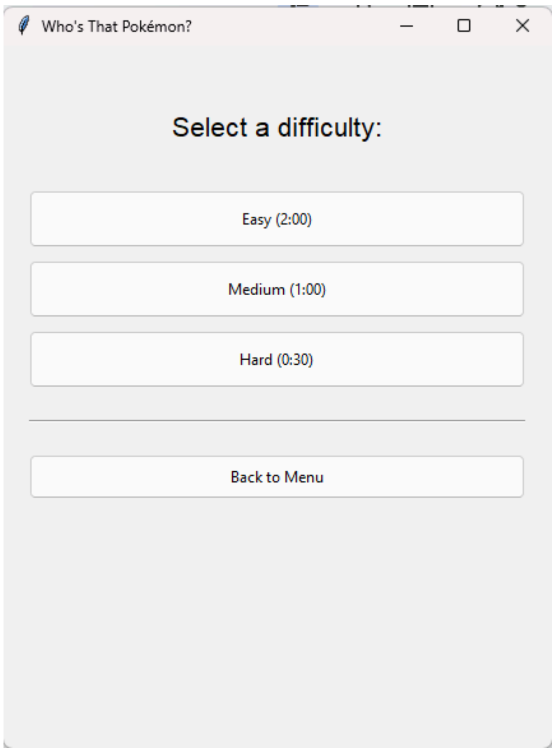
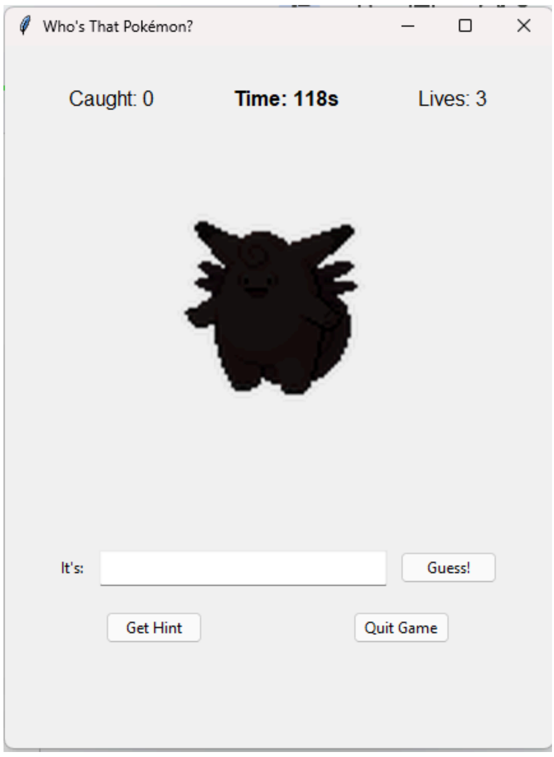
Fix method name and clean up comments in pokeapi.h
Fraxinity authored 5 days ago — Verified — fe59a68

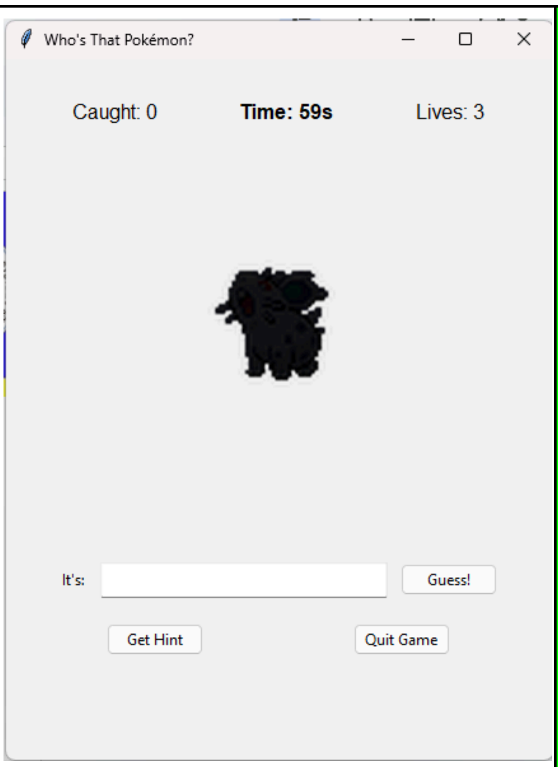
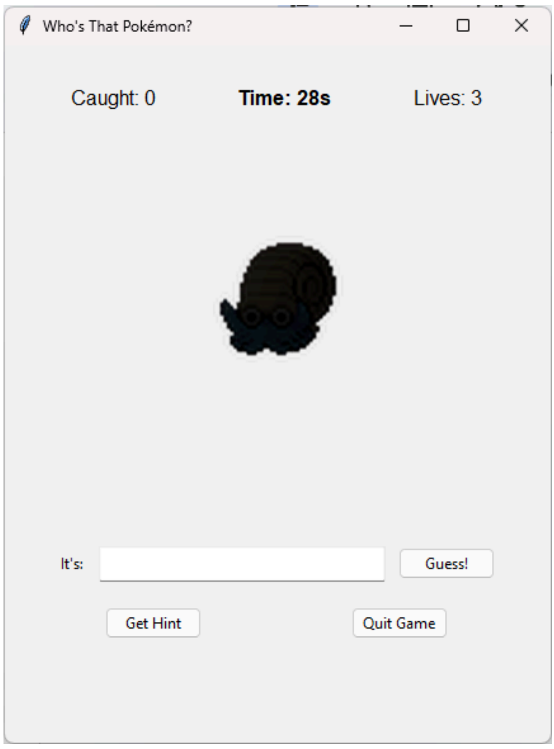
Merge pull request #8 from UPHSL-CCS/pokedex-adam
Fraxinity authored last week — Verified — c229c7b

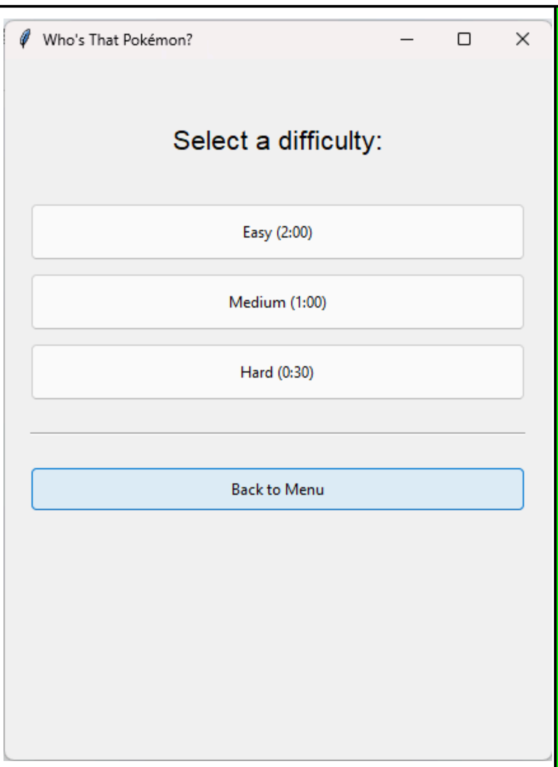
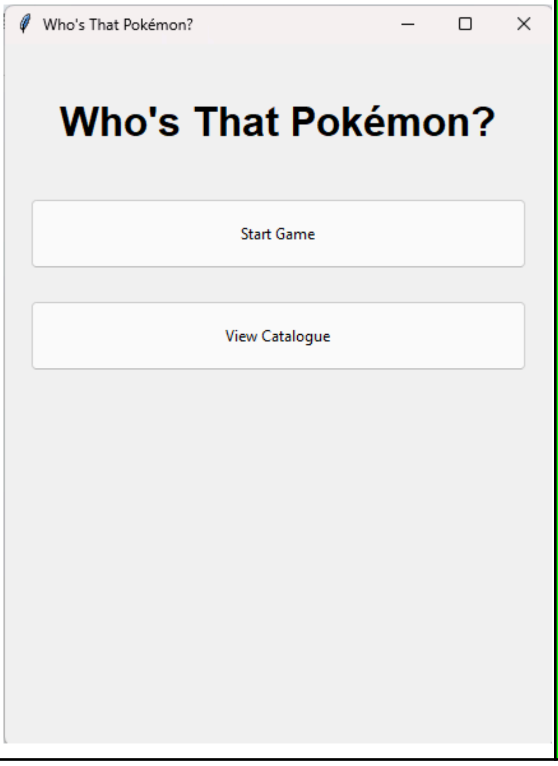
Git commit helps us understand what our team is makingJ or contributing to the system, whether it be a feature or bug fixes. This also shows accountability if ever something goes wrong within the system. This is why we work on branches to minimize those mistakes and contribute to the system more safely.

Testing and Evaluation

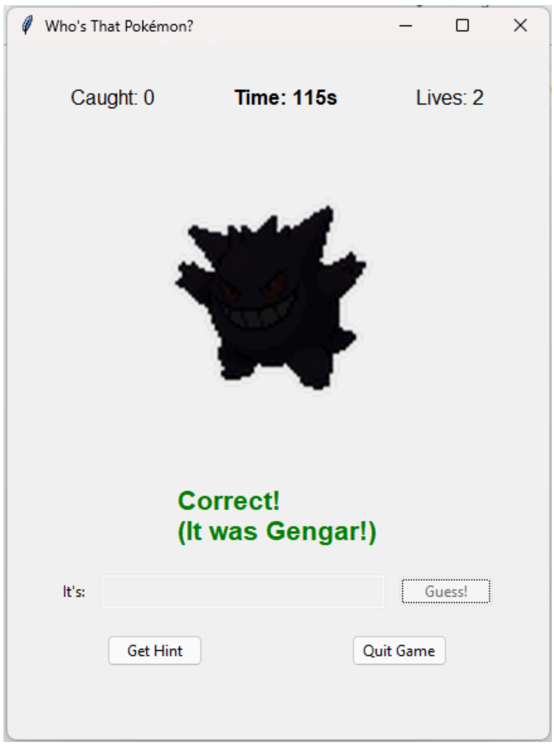
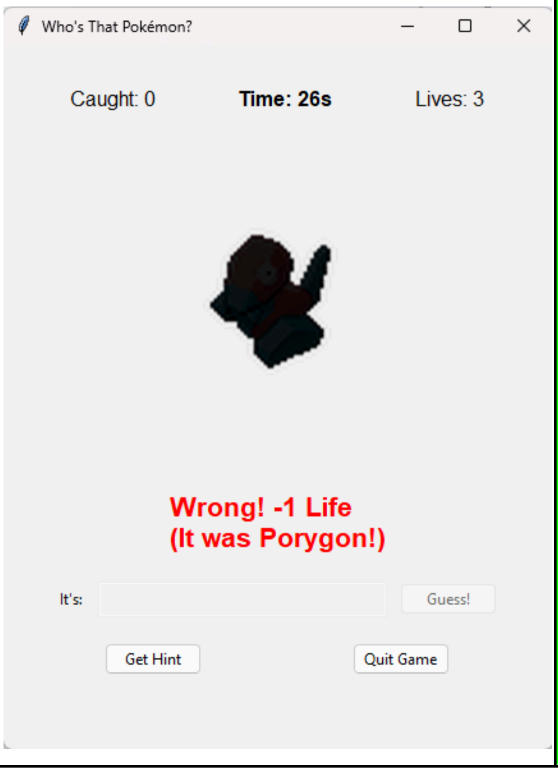
Test Cases Used

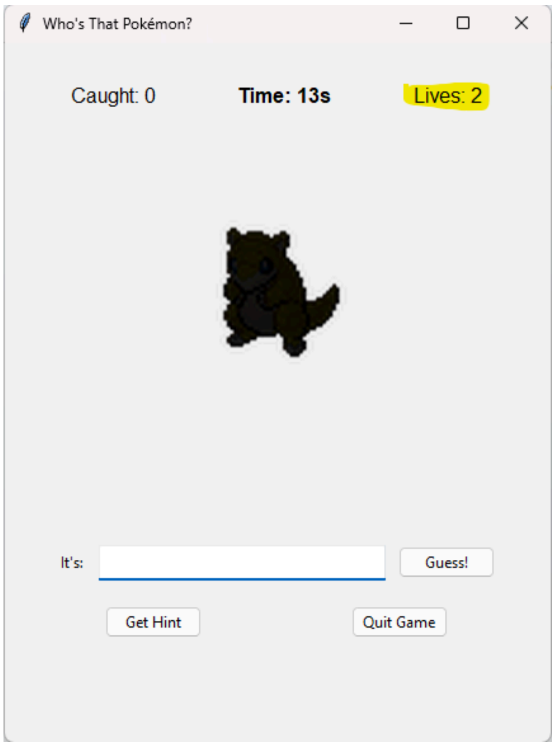
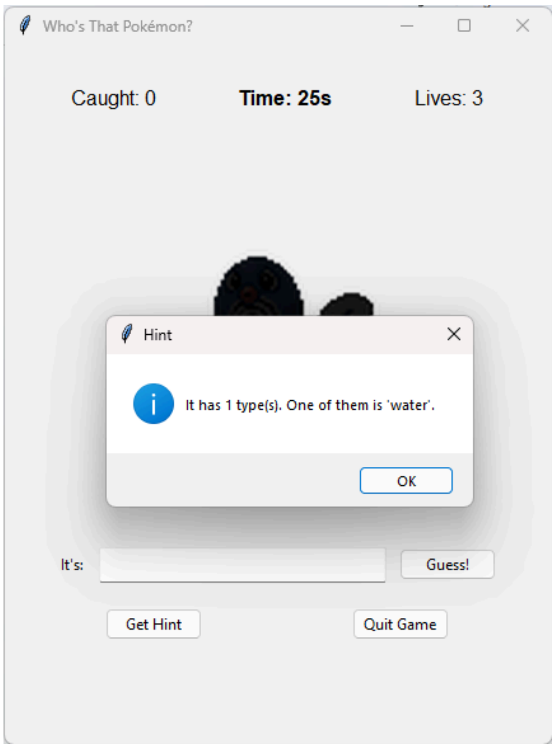
Feature	User Interaction	Expected Result (Pass Criteria)	Proof of Technology	Status
Start Game (Selection)	1. Click "Start Game" on the StartMenuScreen.	The app switches to the DifficultyScreen, displaying the options: Easy, Medium, Hard, and Back to Menu.	 Who's That Pokémon? Select a difficulty: Easy (2:00) Medium (1:00) Hard (0:30) Back to Menu	Passed
Start Game - Easy	1. Click "Start Game" on the StartMenuScreen. 2. Click "Easy (2:00)" on the DifficultyScreen.	The app switches to the GameScreen. The Timer starts at 120s. Lives are set to 3. A silhouette image appears.	 Who's That Pokémon? Caught: 0 Time: 118s Lives: 3 It's: Guess! Get Hint Quit Game	Passed

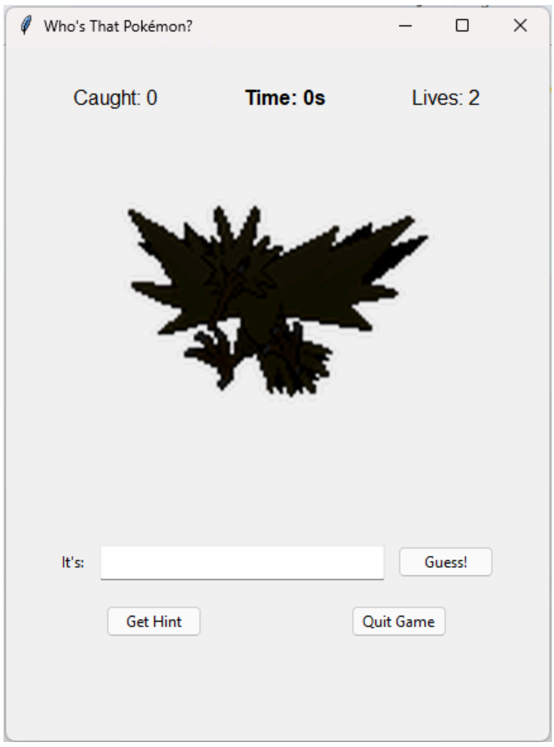
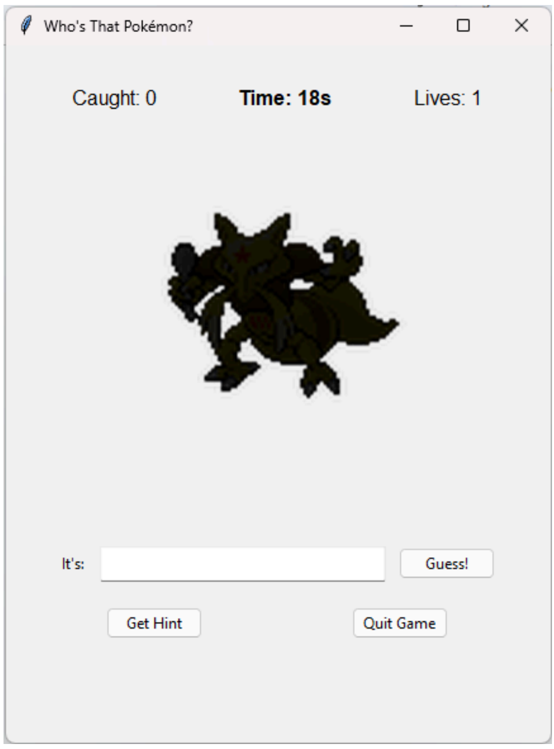
Start Game - Medium	1. Click "Start Game" on the StartMenuScreen. 2. Click "Medium (1:00)" on the DifficultyScreen.	The app switches to the GameScreen. The Timer starts at 60s (1:00). Lives are set to 3. A silhouette image appears.	 Who's That Pokémon? Caught: 0 **Time: 59s** Lives: 3 	Passed
Start Game - Hard	1. Click "Start Game" on the StartMenuScreen. 2. Click "Hard (0:30)" on the DifficultyScreen.	The app switches to the GameScreen. The Timer starts at 30s. Lives are set to 3. A silhouette image appears.	 Who's That Pokémon? Caught: 0 **Time: 28s** Lives: 3 	Passed

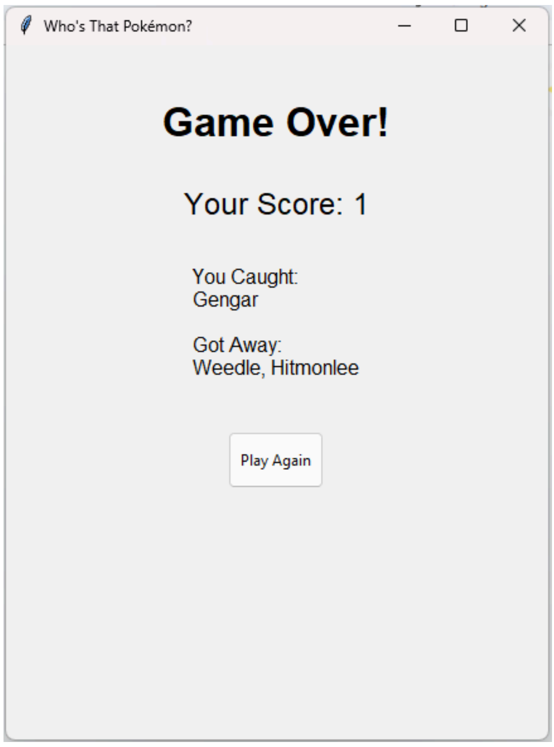
Back to Menu	On the DifficultyScreen, click "Back to Menu".	The app switches back to the StartMenuScreen.	 Who's That Pokémon? Select a difficulty: Easy (2:00) Medium (1:00) Hard (0:30) Back to Menu  Who's That Pokémon? Who's That Pokémon? Start Game View Catalogue	Passed

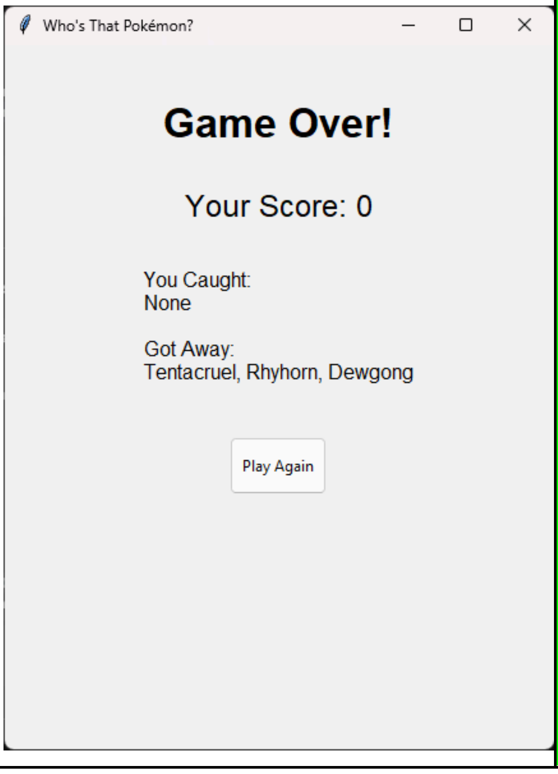
Feature / Action	**User Interaction**	**Expected Result (Pass Criteria)**	**Proof of Technology**	**Status**

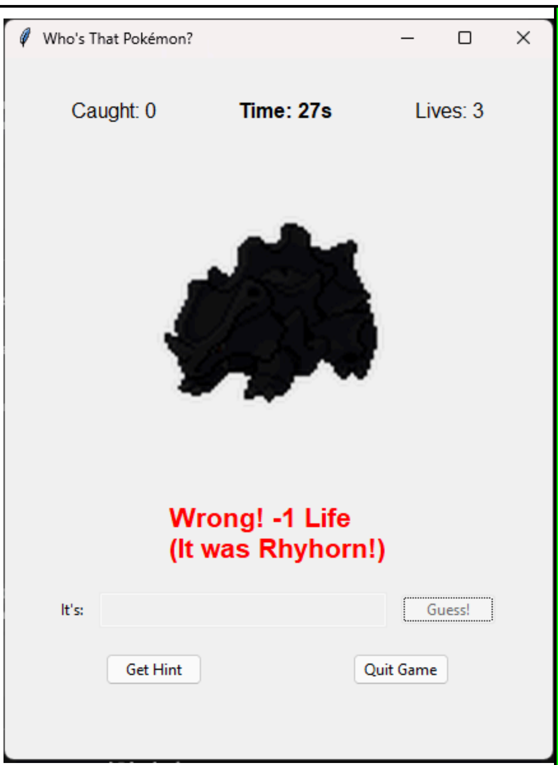
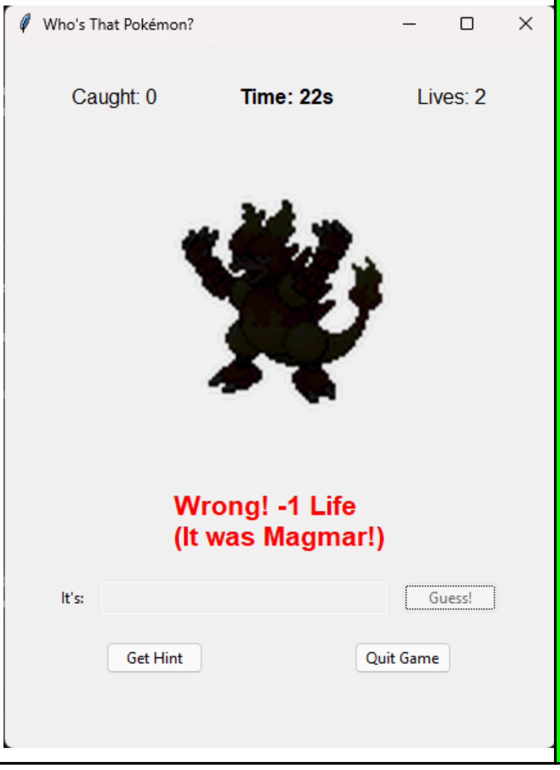
Correct Guess	Type the correct Pokémon name (e.g., 'Pikachu') and click "Guess!"	**Result Label displays "Correct!" in green.** **Score increments by 1.** **After 2 seconds, a new silhouette loads, and the timer restarts.**	 Who's That Pokémon? Caught: 0 Time: 115s Lives: 2 Correct! (It was Gengar!) It's: 	Passed
Wrong Guess	Type an incorrect Pokémon name (e.g., 'Bulbasaur') and click "Guess!"	**Result Label displays "Wrong! -1 Life" in red and reveals the correct name.** **Lives decrease by 1.** **After 2 seconds, a new silhouette loads, and the timer restarts.**	 Who's That Pokémon? Caught: 0 Time: 26s Lives: 3 Wrong! -1 Life (It was Porygon!) It's: 	Passed

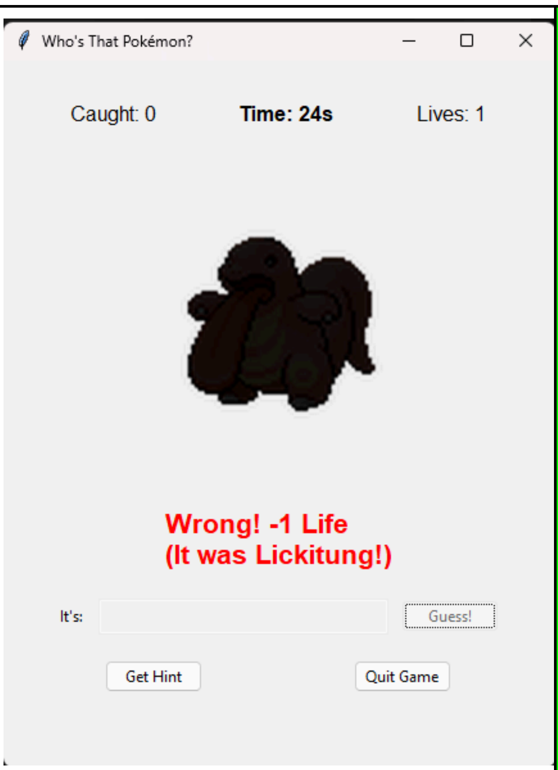
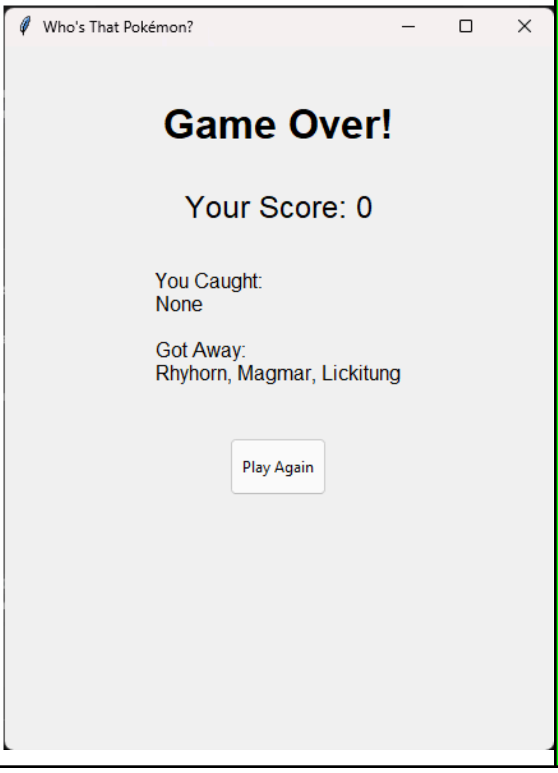
			 Who's That Pokémon? Caught: 0 Time: 13s Lives: 2 It's: Guess! Get Hint Quit Game	
Hint Feature	Click the "Get Hint" button.	A message box appears with a hint (e.g., "It has 2 type(s). One of them is 'Grass'."). The timer does not stop.	 Who's That Pokémon? Caught: 0 Time: 25s Lives: 3 Hint It has 1 type(s). One of them is 'water'. OK It's: Guess! Get Hint Quit Game	Passed

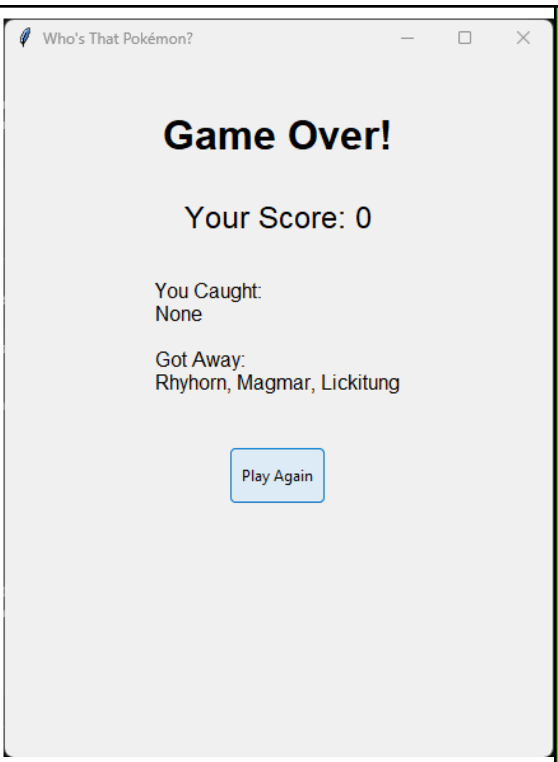
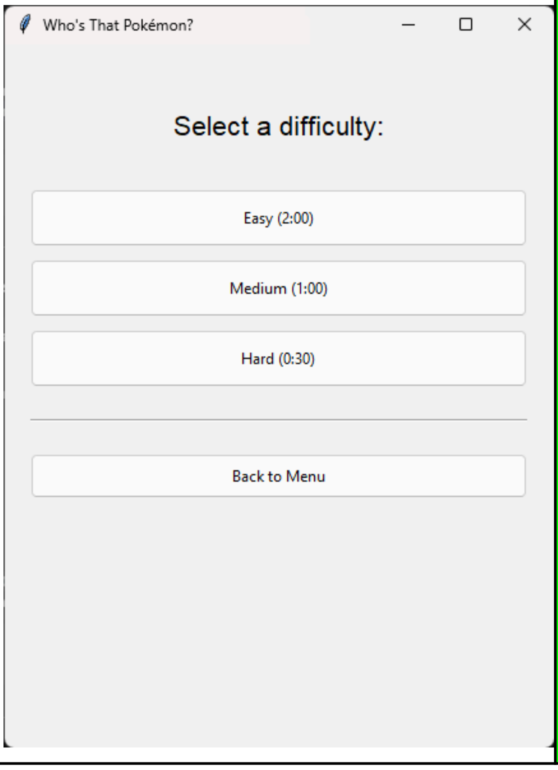
Time Out	Let the timer run down to 0s.	**Result Label displays "Time's Up!" in red and reveals the correct name.** **Lives decrease by 1.** **After 2 seconds, a new silhouette loads, and the timer restarts.**	 Caught: 0 **Time: 0s** Lives: 2 It's:  Caught: 0 **Time: 18s** Lives: 1 It's: 	Passed

End Game - Quit	Click the "Quit Game" button on the GameScreen.	The app immediately switches to the GameOverScreen and displays the final score and results.	 Game Over! Your Score: 1 You Caught: Gengar Got Away: Weedle, Hitmonlee 	Passed

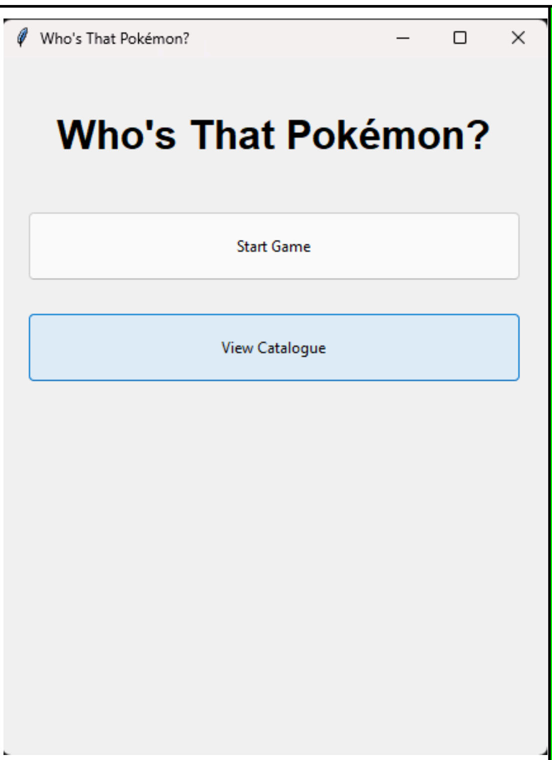
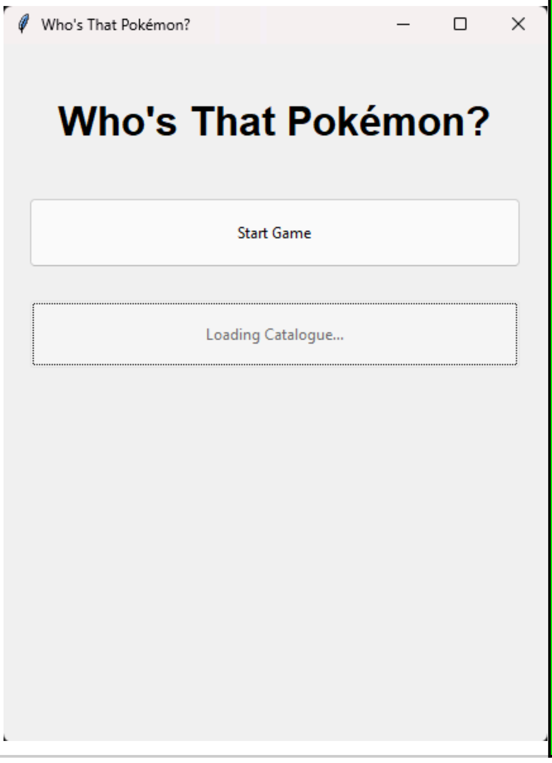
Feature / Action	User Interaction	Expected Result (Pass Criteria)	Proof of Technology	Status
Game Over by Lives	Get 3 wrong guesses or 3 timeouts in a row.	The app switches to the GameOverScreen. The Score label shows the final caught count. The Results label accurately lists the names of Caught and Got Away Pokémon.	 Game Over! Your Score: 0 You Caught: None Got Away: Tentacruel, Rhyhorn, Dewgong 	Passed

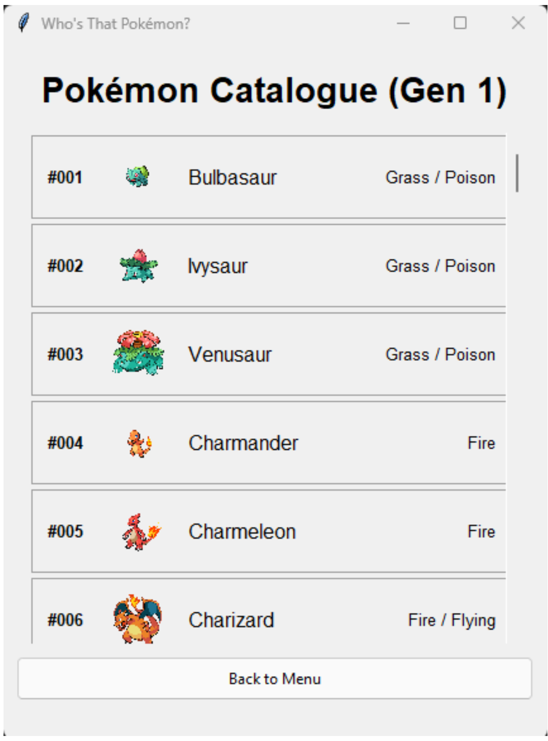
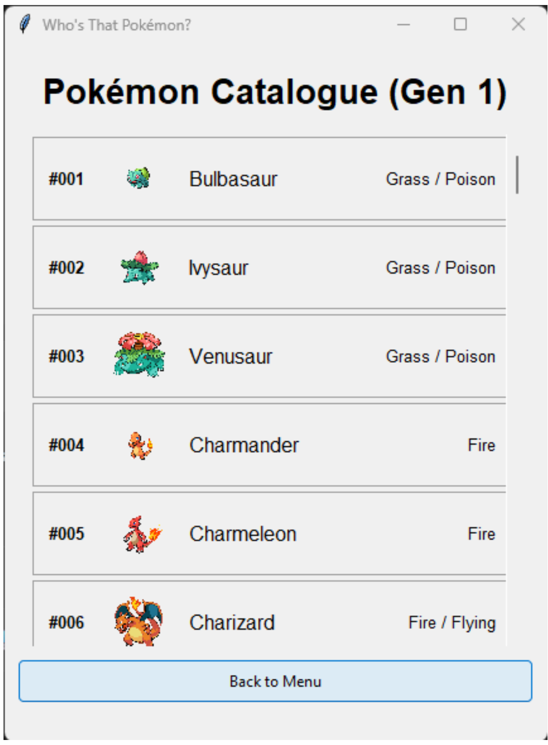
			 Who's That Pokémon? Caught: 0 Time: 27s Lives: 3 Wrong! -1 Life (It was Rhyhorn!) It's:  Who's That Pokémon? Caught: 0 Time: 22s Lives: 2 Wrong! -1 Life (It was Magmar!) It's: 	

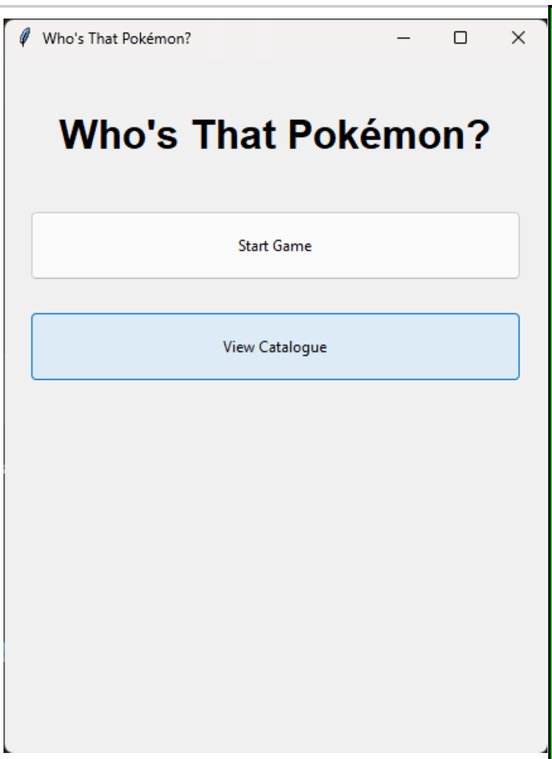
			 Caught: 0 **Time: 24s** Lives: 1 **Wrong! -1 Life** **(It was Lickitung!)**  **Game Over!** Your Score: 0 You Caught: None Got Away: Rhyhorn, Magmar, Lickitung 	🟩

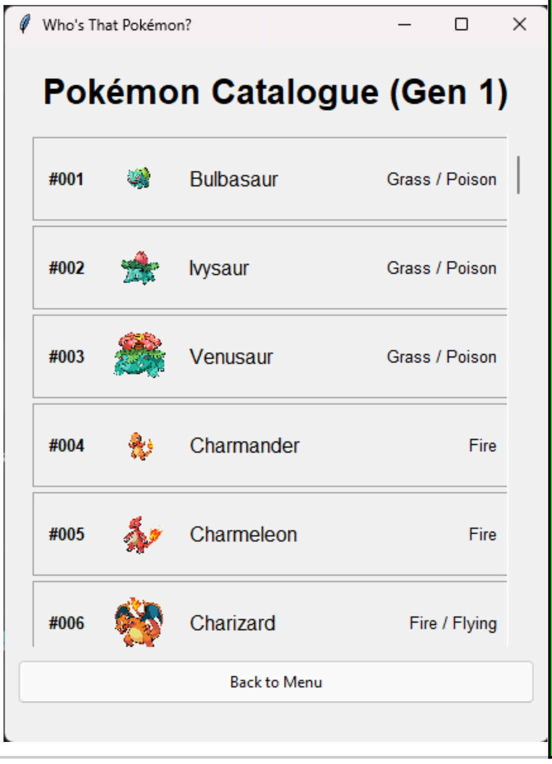
Restart Game	On GameOverScreen, click "Play Again".	The app switches back to the DifficultyScreen.	 Who's That Pokémon? **Game Over!** Your Score: 0 You Caught: None Got Away: Rhyhorn, Magmar, Lickitung  Who's That Pokémon? Select a difficulty: 	Passed

Feature / Action	**User Interaction**	**Expected Result (Pass Criteria)**	**Proof of Technology**	**Status**

Initial Catalogue Load	1. Click "View Catalogue" on the StartMenuScreen.	While loading, the main UI remains responsive. Upon success, the app switches to CatalogueScreen. The screen displays 151 entries, each with: ID (#001), Sprite Image, Name, and Types (e.g., Fire / Flying).	 Who's That Pokémon? Who's That Pokémon? Start Game View Catalogue  Who's That Pokémon? Who's That Pokémon? Start Game Loading Catalogue...	Passed

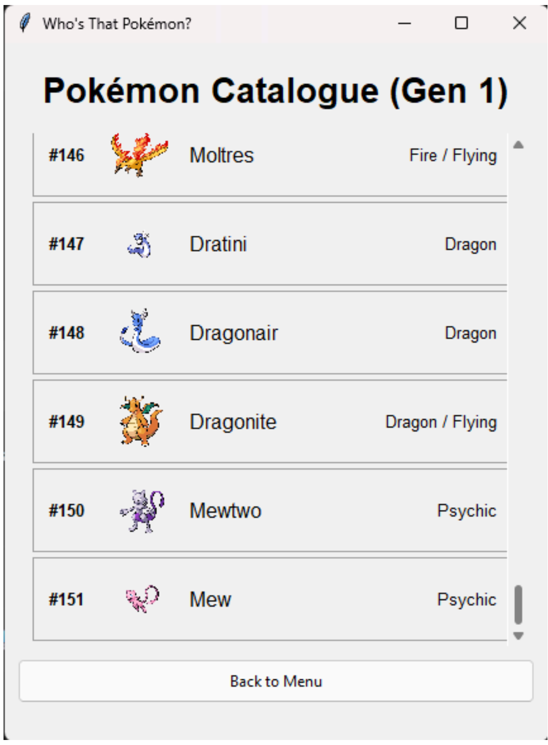
			 Who's That Pokémon? Pokémon Catalogue (Gen 1) #001 Bulbasaur Grass / Poison #002 Ivysaur Grass / Poison #003 Venusaur Grass / Poison #004 Charmander Fire #005 Charmeleon Fire #006 Charizard Fire / Flying Back to Menu	
Subsequent View	1. Click "Back to Menu" from CatalogueScreen. 2. Click "View Catalogue" again.	The app instantly switches to the CatalogueScreen without the "Loading..." delay.	 Who's That Pokémon? Pokémon Catalogue (Gen 1) #001 Bulbasaur Grass / Poison #002 Ivysaur Grass / Poison #003 Venusaur Grass / Poison #004 Charmander Fire #005 Charmeleon Fire #006 Charizard Fire / Flying Back to Menu	Passed





Pokémon Catalogue (Gen 1)

#001		Bulbasaur	Grass / Poison
#002		Ivysaur	Grass / Poison
#003		Venusaur	Grass / Poison
#004		Charmander	Fire
#005		Charmeleon	Fire
#006		Charizard	Fire / Flying

Scroll Functionality	Scroll down the list of 151 entries.	The user can smoothly scroll through the entire list of Pokémon entries.	 Who's That Pokémon? Pokémon Catalogue (Gen 1) #146 Moltres Fire / Flying #147 Dratini Dragon #148 Dragonair Dragon #149 Dragonite Dragon / Flying #150 Mewtwo Psychic #151 Mew Psychic Back to Menu	Passed
Error Handling (Simulated)	(Developer simulates an API failure during load)	A message box appears showing the "Catalogue Error" message. The app returns to the StartMenuScreen. The "View Catalogue" button is reset to its normal state.		Passed

Discussion of Results, Issues, or Limitations

Ethical and Professional Reflection

Each group must answer the following questions:

- **How did your team ensure ethical collaboration (no plagiarism, fair contribution)?**

We utilized Git and GitHub to maintain a transparent history of contributions. Each member's work is verifiable through commit logs, ensuring accountability. To avoid plagiarism, we strictly adhered to Open Source licensing by properly importing and attributing external libraries rather than copying source code directly. We also divided the project using a Modular Architecture UI, Logic, API, which naturally enforced fair division of labor without overlapping code. We also contributed to each other's documentation and user interface development, balancing the workload across the different languages (Python and C++) used by the group.

- **How does your system ensure data privacy (if applicable) and responsible programming?**

While our system does not store personal information, we practiced Data Minimization by keeping game states in ephemeral memory rather than writing to persistent storage. Regarding responsible programming, we implemented API stewardship. Instead of spamming the PokéAPI with requests on every frame update, we fetch data only when necessary and cache results where possible to respect the provider's server resources and rate limits

- **What lessons can you apply from professional practice and version control ethics?**

We applied the industry standard of making small, descriptive changes rather than bulk, vague uploads and making each draft branches (e.g. draft-angela, draft-aninang, and draft-mickz). This ensures that if a bug is introduced, we can easily rollback without breaking the entire build.

Independent Learning Component

Each member must write a short paragraph (3–5 sentences) describing:

- What new concept or tool they independently learned (e.g., Git branching, threading, or modular architecture).
- How it improved their contribution.

Aninang, Kelvin Adam

Learning API through this project brings me so much insight on how to handle and call data from a different source. Handling API is a tricky but as long as the document exists, I could understand how it functions and where to use them for our system. I also learned how concurrency not only exists within the data but for UI. Worker threads do help a lot when managing 151 pokemons for CPU processing.

Learning these concept and tools, I have utilized and applied this knowledge to our 2nd project called Pokedex. These knowledge proves working with different languages can sometimes parallel because I applied the same of concurrency with worker threads to not only the 151 pokemons but to the quiz to avoid UI freezing. With these, the system would run smoothly just like the pokedex

Cabanes, Angela

This project really is a breakdown of every lesson taught in programming. I learned about the difference between syntax and systematic errors and how to avoid them, which is achieved by simply understanding the logic of your program better and not forgetting semicolons and such. I also learned about installing extensions like linters, the use of tokens, variable scopes, and subprograms (modularity and abstraction). The subprograms are represented by the multiple program files inside our repo, and the abstraction is the use of the API itself. I also used concurrency, as this program fetches data from the API and runs the time at the same time without crashing the whole app. Adding the feature "view catalog" took a lot of time even though the said feature may be simple, as the app kept crashing. With that, I also learned to use threading again so you can wait for everything to wait on the catalog and still have the app running. Doing this activity really is tricky, but I am also learning as I go along.

Other than that, I also learned how to do Git branching, as I am new to Git. Everything I have learned, I have applied to the internship and project I am currently working on. I learned how to use them in a way that is useful and would expose the contributions you made inside the project. Overall, as stressful as it may be, I really enjoyed doing it. I had fun learning.

Jornales, Michaela

I learned how to implement Python Threading in the Tkinter GUI application. And then using threading module to run background tasks parallel to the main application loop.

The Pokémon API helped the system by fetching data from the internet but running this on the main thread caused the entire application window to freeze until the download was complete.

References

Anderson, J. (2024, August 5). *An intro to threading in Python*. https://realpython.com/intro-to-python-threading/

Andy's Tech Tutorials. (2022, September 21). *Pokemon API Tutorial | For Beginners* [Video]. YouTube. https://www.youtube.com/watch?v=V_cMYN3jAjg

Jeremiah, O. (2024, November 24). *Mastering Python APIs: A comprehensive guide to building and using APIs in Python*. DataCamp. Mastering Python APIs: A Comprehensive Guide to Building and Using APIs in Python

Splunk. (n.d.). What is concurrency? Retrieved November 25, 2025, from https://www.splunk.com/en_us/blog/learn/concurrency.html

Perez, S. (2019, November 7). Fetching them all: Poke API. Medium. https://medium.com/@sergio13prez/fetching-them-all-poke-api-62ca580981a2

PokéAPI. (n.d.). PokéAPI: The RESTful Pokémon API. Retrieved November 2025, from https://pokeapi.co/