Samuel Oseguera

5/13/2025

CPSC-25-10103

Final Project Report

Application Explanation:

My final programming project for this course is a Pokémon-themed Gacha Game pull simulator. The application itself focuses solely on the summoning part that acts as a core mechanic. Players playing real gacha games can exchange in-game currency for a chance to pull characters of varying rarity to add to their teams. The terminal-based application I created allows the user to play that for free. The user is presented with the option to either do single pulls or 10x pulls. These are reminiscent of the system real gacha games use. The user, depending on the choice of pulls they make, is then given the name and rarity of the Pokémon they received via a message. The selection pool for the Pokémon that a user can obtain comes from a CSV created using a Pokémon database website. The Pokémon are currently limited to the original Gen1 list. This application can be entertaining because of its wide selection pull and rarity-based chances. The player can continuously attempt to obtain the Pokémon they want over and over. There are 151 current Pokémon that are a part of that Gen1 list.

Here is an example of all features that could be outputted by a player:

```
Welcome to the Pokémon Gacha Pull Simulator!
---------------------------------------------
Choose pull type:
1. Single Pull
2. 10x Pull

2
Performing 10x pull...
Pull 1: 3-star Pokémon: Dugtrio
Pull 2: 3-star Pokémon: Venonat
Pull 3: 4-star Pokémon: Pidgeot
Pull 4: 3-star Pokémon: Clefable
Pull 5: 3-star Pokémon: Persian
Pull 6: 3-star Pokémon: Jolteon
Pull 7: 3-star Pokémon: Clefable
Pull 8: 3-star Pokémon: Clefable
Pull 9: 3-star Pokémon: Tentacruel
Pull 10: 3-star Pokémon: Zubat
---------------------------------------------
Pull again? (y/n): y
Choose pull type:
1. Single Pull
2. 10x Pull
1
You pulled a 3-star Pokémon: Omanyte
---------------------------------------------
Pull again? (y/n): n
Thanks for playing!
```

Algorithms and Flow Charts:

Algorithm 1: CSV Parsing and Rarity Assignment

```cpp
    // Open the CSV file that contains the Pokémon data
    std::ifstream file("gen1_pokemon.csv");

    if (!file) {
        std::cerr << "Error: Could not open the file.\n";
        return 1;
    }

    // Read the file line by line and assign rarity based on position
    std::string line;
    std::getline(file, line);

    int count = 0;
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string number, name;
        std::getline(ss, number, ','); // Ignores the Pokémon's pokedex number
        std::getline(ss, name, ',');   // Grabs the Pokémon name instead

        // Assigns rarity for the Pokémon based on CSV list position
        int rarity;
        if (count < 10) {
            rarity = 5;
        } else if (count < 30) {
            rarity = 4;
        } else {
            rarity = 3;
        }

        pokemons.push_back({name, rarity});
        ++count;
    }

    file.close();
```

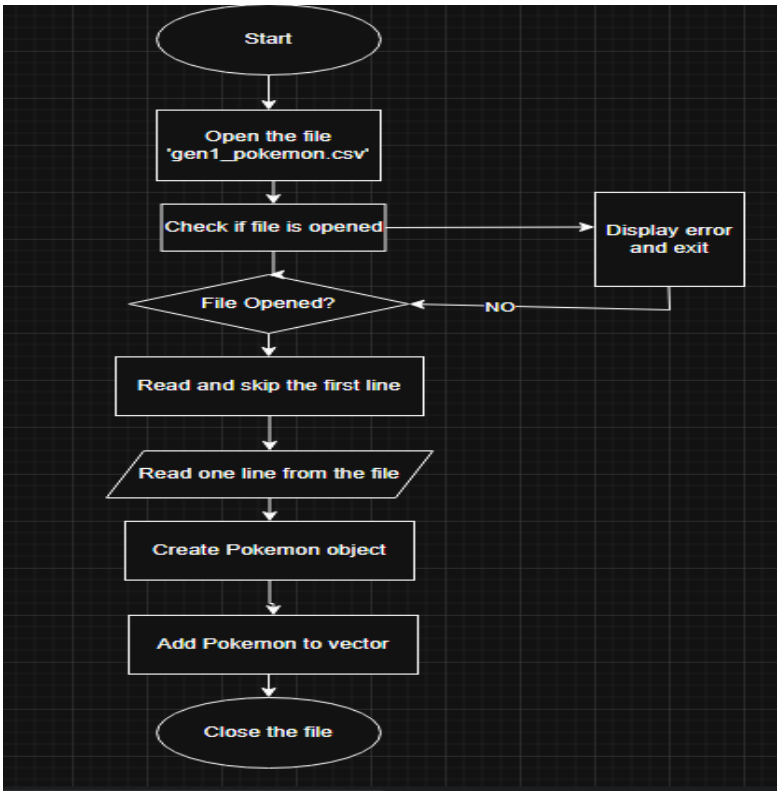
The first algorithm out of the three in this report is going to be the CSV file parsing algorithm. This algorithm reads Pokémon data from the CSV file (gen1_pokemon.csv) I created using pokemondb.net. It extracts the names of all 151 Generation 1 Pokémon and

assigns each one a rarity level from 3-star, 4-star, or 5-star based on their position in the CSV file. The Pokémon read from this file are then stored in a vector of Pokémon structs for use in the gacha simulator. I did ask ChatGPT for help when it comes to the rarity percentage values. I fed it some of the most popular gacha games on the market and told it to give me values that would match the current gacha landscape. It's an integral part of the entire program as it sets the foundation for the core features.

The following is the Series Steps for the algorithm:

1. Opens the CSV file using std::ifstream.
2. Checks if the file opened successfully. If it doesn't, an error message will print, and the algorithm will exit.
3. Reads and discards the header row.
4. Initializes a counter variable to track how many Pokémon have been processed.
5. Each line in the file:
    - Uses a stringstream to split the line into columns
    - Skips the first column
    - Reads the second column, which contains the Pokémon's name
    - Assigns rarities based on counter value
    - Creates a Pokémon struct with the name and rarity
    - Adds the struct to the pokemons vector
    - Increments the counter
6. Closes the file after reading all the data.

I'm also including a flowchart that I created for each to help visualize the control flow for this algorithm.



The Big O Time Complexity of this program comes out to be O(n). The n here corresponds to the number of Pokémon in the CSV file. As mentioned before, the total number of Pokémon in Gen 1 comes out to 151. So, each Pokémon is processed once.

Algorithm 2: Gacha Pull Simulation

```cpp
Pokemon gachaPull(const std::vector<Pokemon>& pokemons) {
    // Set up random number generation to simulate gacha roll (0 to 100)
    // ChatGPT was used to help understand how to implement std::random_device and std::uniform_real_distribution
    // Citation: OpenAI. (2025). ChatGPT [Large language model]. https://chat.openai.com
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(0.0, 100.0);

    double roll = dis(gen); // Rolls from (0-100)
    int rarity = 3; // Sets default roll to 3-star Pokemon

    //Assigns rarity based on typical gacha chances

    if (roll <= 2.0) {
        rarity = 5;
    } else if (roll <= 12.0) {
        rarity = 4;
    }

    // Filter Pokémon by the selected rarity
    std::vector<Pokemon> filtered;
    for (const auto& p : pokemons) {
        if (p.rarity == rarity) {
            filtered.push_back(p);
        }
    }

    // Randomly select one Pokémon from the filtered list
    std::uniform_int_distribution<> pick(0, filtered.size() - 1);
    return filtered[pick(gen)];
}

```

The second algorithm in this report is going to be the gacha pull algorithm, which is implemented using the gachaPull() function. This algorithm simulates a randomized pull that chooses from the CSV file's selection pool of Pokémon and their rarity level assignments. It assigns those rarity levels using the random number generator and then filters the Pokémon list to match the selected rarity. After doing so, it randomly selects a Pokémon from the filtered list and returns it. This algorithm matches the real gacha games systems for giving out random characters to the players.

I did obtain help from ChatGPT for this algorithm (the code is cited to show the parts where help was given). As mentioned before, the number generators and percentage values were codes that ChatGPT guided me through. I asked it to give me something like the current market gacha games. I listed off a list of the popular games, as mentioned before, and that is where I was able to get help from ChatGPT with this algorithm.

The following is the Series Steps for the algorithm:

1. Generates a random number between 0.0 and 100.0 using std::uniform_real_distribution
2. Based on the number rolled
    - If the number is <= 2.0 the assigned rarity = 5-star Pokémon
    - If the number is <= 12.0 the assigned rarity = 4-star Pokémon
    - If none of these conditions are met the assigned rarity becomes = 3-star Pokémon
3. Creates an empty vector to store the filtered Pokémon
4. Loops through the full Pokémon list

5. Generates a new random index from 0 to the size of the filtered list
6. Returns the Pokémon at the randomly selected index

The Big O Time complexity for this algorithm is broken down into different sections. The filtering step of the gacha pull algorithm has a Big O time complexity of O(n) with the n representing the total number of Pokémon, very similar to the parsing algorithm. The random selection part of this algorithm has a Big O time complexity of O(1). And the total, overall time complexity of the algorithm ends at O(n). It maintains pretty efficient because of the list size only being 151 while matching real gacha mechanics.

Algorithm 3: 10x Pull Loop

```
do {
    // Gives the user the choice to do a single pull or a 10x pull
    std::cout << "Choose pull type:\n";
    std::cout << "1. Single Pull\n"; // Enter 1 for a single pull
    std::cout << "2. 10x Pull\n"; // Enter 2 for a 10x pull
    int pullChoice;
    std::cin >> pullChoice;

    if (pullChoice == 1) {
        //Single pull output
        Pokemon pulled = gachaPull(pokemons);
        std::cout << "You pulled a " << pulled.rarity << "-star Pokémon: " << pulled.name << "\n";
    } else if (pullChoice == 2) {
        // 10x pull output
        std::cout << "Performing 10x pull...\n";
        for (int i = 0; i < 10; ++i) {
            Pokemon pulled = gachaPull(pokemons);
            std::cout << "Pull " << (i + 1) << ": " << pulled.rarity << "-star Pokémon: " << pulled.name << "\n";
        }
    }
```

The final and third algorithm in this report is the 10x pull feature found in the main program. This algorithm allows the user to choose to pull 10 Pokémon at once during the simulation instead of doing single pulls. This algorithm uses a loop to call gachaPull() function repeatedly and displays the results to the user. It simulates a common feature in gacha-based games where users are given the choice to perform multiple pulls at once for efficiency's sake. These are often accompanied with bonus rewards to incentive players to save up for these instead. Though there are none of those in my current project, it could be a worthwhile addition to incorporate at a later date. For example, one gacha game I play gives the player a free 10x pull after they've chosen the 10x pull option 3 times.

The following is the Series Steps for the algorithm:

1. Asks the user if they want to do a single pull or a 10x pull
2. Depending on the user's choice, if they choose to do a10x pull then:
    - Prints a message indicating that the 10x pull is starting

- Initializes a for loop that iterates 10 times
- Each iteration:
    - Calls the gachaPull() function to get a random Pokémon
    - Displays the pull number, the rarity, and the pokemons name

3. Once all 10 pulls are done, it will ask the user if they want to pull again

The Big O time complexity for this algorithm is going to be O(n) because it is a simplification of the O(10n) that comes from the 10 loops.

Data Structures

Struct Data Type: Pokémon

The first data structure used in this project is a C++ struct named Pokémon. I created this struct to combine related data (name and rarity) into a single unit. This makes it much easier to manage each Pokémon as an object during the pull simulation, rather than managing separate arrays or variables for names and rarities. Each Pokémon parsed from the CSV file is converted into a Pokémon struct and stored in a vector. This struct is also passed to and returned from the gachaPull() function, making it a core part of the program's design.

Dynamic Array: std::vector

The second data structure I am explaining in this report is the std::vector dynamic array. The purpose of this data structure is to store all Pokémon from the CSV file and later filter them based on their assigned rarity. I chose a vector because it supports dynamic resizing and provides efficient random access, making it ideal for storing the list of Pokémon and selecting from it at random. In main.cpp, the vector holds all parsed Pokémon. In gacha.cpp, a second filtered vector is created from the original list to contain only Pokémon that match a given rarity.

Text Storage: std::string

The final data structure explained in this project is going to be the std::string text storage data structure. The purpose of this data structure is to store Pokémon names and user

inputs. I chose std::string because it's standard for handling and manipulating text. The program requires it to be able to read and print all Pokémon names listed in the gen1_pokemon.csv file. It's used in the program to store the Pokémon name inside each struct, to parse the CSV file input line by line, and for displaying results and messages to the user.

Step where an opportunity was found:

I'd say the step in my project in which I found a good opportunity to use would probably have been related to the CSV file. Having a database that could give me so much information and statistics pertaining to every Pokémon possible was very useful, even with me not utilizing it to the max. One of the points I planned to use in the project was the ability to output the statistics of Pokémon (like types, attributes, pokedex numbers) to the user after every pull. I didn't manage to get around to completing it for this project's current state but having that available is so useful.

Step where an error was found:

Going back to the CSV file portion again, one error that constantly occurred when working on the project was the creation of the file. I originally planned to have a very limited selection pool based off a list I had found elsewhere, but it was very annoying and finicky to work with my own file. Specifically, the formatting of the Pokémon information in relation to the output given to the user after pulls. The output kept giving out only the Pokedex number and it wouldn't give out the name of the Pokémon. It didn't look so good having an output that looked like this: "You pulled the 5-star Pokémon: 5". Especially without the ability to know which Pokémon that number pertains to. All I did was add a small tweak to the parsing algorithm's code that allowed it to skip columns. The output then fixed itself to come out like it does in the final product.

Future changes I would add to the project:

The biggest feature that comes to mind, and I had listed as a ceiling end-goal for this project, is to add graphics to this project. A simulator like this would have so much more added entertainment with visuals for the player to see. My idea was to have two buttons for each of the pull number amount types with a Poke ball in the middle. The player would choose whichever of the two they would like, and a pixel sprite of the Pokémon would appear. I want to mess around with it during the summer and see if I can incorporate it.

Another feature would be an ordered pull history that would pop up for the user at the very end, letting them know the amount of 5-star, 4-star, and 3-star Pokémon they got during the simulation session. And the last thing that I would think to currently add would be the ability to search more in-depth information of all the Pokémon prior to pulling. That way they can see their favorite Pokémon's rarity assignment and statistics and attempt to pull for them.

Bibliography:

ChatGPT Citation (APA Style):

OpenAI. (2025). *ChatGPT* [Large language model]. https://chat.openai.com

I used ChatGPT to assist with setting up the random number generation logic for my gacha pull system. Specifically, it helped me understand how to use std::random_device, std::mt19937, and std::uniform_real_distribution together to simulate a realistic probability-based pull. I also used ChatGPT to determine appropriate rarity percentage values based on mechanics from common gacha game