Xiyu Fan
CS 5001
Final Project Design Document Updated Version

I. Basic Design Concept

My project aims at managing the park and park facility data, finding parks with certain conditions on a map, and analyzing the relationship between a neighbourhood and park facilities inside it. It's a program that mainly designed for urban planners or urban planning gamers to assess the current development of park facilities in different parks and neighbourhood areas and to decide how to maintain these facilities to meet the public's recreation needs, while its filter function can also be helpful for the general public to decide which park they would like to go or which neighbourhood is the best to move into depending on their outdoor leisure needs.

II. Classes Design

Correspondingly, I created two classes named Park and Facility in model_1_park_class.py and model_2_facility_class.py. The Park class is a module that provides functionalities to manage a park model in Vancouver City. It is responsible for reading the name of a park, the name of the neighbourhood that this park is located, the park' area, and the park's latitude and longitude. Its main features include checking whether the park belongs to a certain neighbourhood, expanding the park's area, and decreasing the park's area. The Facility class is a module that provides functionalities to manage a park facility model in Vancouver City. It is responsible for reading the name of the park that this facility belongs to, the facility type, and its count. The main features of the park facility module include checking if the facility type corresponds to a certain type, checking if the facility count is greater than a certain count, and increasing and decreasing the facility count.

Compared with Milestone 1, I added two attributes, the park's latitude and longitude, and one method, is_neighbourhood, to the Park class. Correspondingly, I also added two methods, is_type, and is_count to the Facility class. With the attributes of latitude and longitude, I created the functionality of finding parks on a map. And the three new methods could help the program filter parks with certain conditions.

III. Data Fetching

To fetch and clean the data, I design three functions for each data source separately and one for cleaning these data together in fetch_data.py. Firstly, to fetch data from the URLs, I design get_parks_csv() and get_facilities_csv(), using the requests.get() method to fetch data into text. Then, by using get_needed_park_data() and get_needed_facility_data(), I try to get the data I needed from the two data resources that contain various useless data for my program, followed by encapsulating these data into two lists of instances by create_park_instances() and create_facility_instances(). Lastly, while I noticed that some of the park hectare data from the data resource is represented as zero, which is unreasonable, I designed remove_invalid_park_instances() to remove invalid park instances whose hectare is zero and their correspond facility instances from the lists created before.

Compared with Milestone 1, I separated these functions into two files, models.fetch_data.py and utils.py, to make sure that the program meets the requirements of the MVC pattern. For the models.fetch_data.py, I kept functions in charge of fetching data (get_parks_csv(), get_facilities_csv(), get_needed_park_data(), get_needed_facility_data()) in it. For the utils.py, I removed the function in charge of creating objects and cleaning invalid objects to it.

IV. Data Structure

As for the usage of data structures, I mainly use lists and dictionaries in model_3_parks_and_facili ties_analysis_functions.py.

Firstly, by create_park_facilitiy_dictionary(), I try to create a dictionary whose keys are park names and values are facility types and counts, representing like {'Park1': [[Pool, 1], [Playground, 2]]}.

Then, I designed the "neighbourhood-facility type-facility count and distribution" whose keys are neighbourhood names and values are dictionaries with facility types as keys and lists of facility types and distribution as values, representing as {'Neighbourhood1': {'Facility1': [3, 0.75], 'Facility2': [1, 0.25]}, 'Neighbourhood2': {'Facility1': [2, 0.5], 'Facility2': [2, 0.5]}}.

Compared with Milestone 1, I streamlined the process of creating dictionaries, making them easier to read and maintain and deleted the facility density dictionary as I addjusted the design direction of this program.

V. Visualization

As for visualization, I used folium to create a map showing parks' location and details and used matplotlib to create pie charts and a bar chart to show the number and distribution of different park facilities in neighbourhoods.

The first visualization functionality, "Find parks and facilities on a map", allows users to explore parks and park facilities within Vancouver City. Users can choose to view all parks in the city, search for parks based on specific conditions, or locate a park by its name. For example, users can check all the parks with at least two playgrounds in the Downtown area. In this step, I iterate all the facility objects and use the method is_type and is_count in the Facility class to check whether they have more than 2 playgrounds, and then iterate all park instances and method is_neighbourhood in the Facility Class to check whether these parks with more than 2 playgrounds are in the target neighbourhood. If both of the answers are Yes, this valid park instance will be put into a list named filtered_park and returned by a function. Then by visualization functions, these qualified parks will be marked orange on the map to show users where they are.

The second visualization functionality, "Analyze the parks and facilities data", allows users to analyze the park facilities' type and count in each neighbourhood. Users could choose to read the result by one neighbourhood or all neighbourhoods, and by charts or by strings.

If the user chooses to read the analysis of one neighbourhood by charts, it will pop up a pie chart showing the count and distribution of park facilities in this neighbourhood. I built the pie chart by a function using Matplotlib. The function will first extract the facility types as labels and their corresponding counts as sizes and their distribution from the facility_dictionary. Then, it constructs label strings by combining facility types, counts, and distribution, and generates the pie chart.

If the user chooses to read the analysis of all neighbourhoods by charts, it will pop up a bar chart showing the count and distribution of park facilities in all the neighbourhoods. In this chart, the x-axis represents the neighborhood names, while the y-axis represents the facility count.

To achieve this, I design a function called display_neighbourhood_facility_bar_chart. It takes a dictionary containing facility information in each neighbourhood as the parameter. Each key in the dictionary represents a neighborhood, and the corresponding value is another dictionary where keys represent different types of facilities, and values are lists containing the count of each facility type. The function first extracts the list of neighborhoods from the facility_dictionary and creates a color map to assign unique colors to each facility type. Next, the function iterates over each neighborhood in the facility_dictionary. For each neighborhood, it retrieves the list of facilities and their corresponding counts. It then creates bars for each facility type within the neighborhood, stacking them vertically to represent the distribution.
Since I haven't done any visualization in the Milestone 1 stage, this part is all brand-new.

VI. Interaction

The user can impact the data analysis mainly in three ways, modifying the park data and the park facility data, choosing filter conditions and finding parks meeting these conditions on a map, and choosing a neighbourhood to read its analysis of park facilities’ type, count, and distribution.
For the first one, the user can increase the area of a park, decrease the area of a park, add a new facility, delete an existing facility, increase the number of a facility, and decrease the number of a facility. All of them can impact the analysis.
For example, if a new park facility is added to a park, let’s say if we build a swimming pool in Stanley Park, considering that there already exists one swimming pool there, the count of Swimming Pool in Stanley Park will increase. This change will be reflected in both the pie chart for the neighbourhood West End and the bar chart for all neighborhoods. The size of the Swimming Pool section in the pie chart for West End will increase, indicating a higher count for that facility type, while the height of the corresponding bar in the bar chart will also increase, representing a higher facility count in that neighborhood.
Conversely, if a park facility is removed from a park or its count is decreased, the corresponding facility type will disappear or have a lower count in the charts, potentially shifting the distribution of facility types within each neighborhood. If we deleted all the Swimming Pools in Stanley Park, its corresponding section in the pie chart for West End would decrease totally disappear, and the height of the corresponding bar in the bar chart would also decrease or disappear, depending on the total number of Swimming Pools in West End.
As the following two functionalities are explained before, I wouldn’t introduce them again here.
Compared to Milestone 1, I deleted the functionality of adding or deleting a park since it would be hard to achieve the functionality I designed before due to my ability limit now.

VII. Upadate

According to feedback received from Milestone 1, I have updated my program in the following ways:

- A clearer driver function
  broke the main function down to several helper functions, making it clear to read.
- A better MVC structure
  I imporved the structure of the program, moving functions shouldn’t have been in Views files to the Models files.
- Using constants

I used constants instead of magic numbers.

- Completed the flowchart

  I completed the flowchart of methods in classes.

QRLs:

Park:

https://opendata.vancouver.ca/explore/dataset/parks/table/?dataChart=eyJxdWVyaWVzIjpbeyJjaGFydHMiOlt7InR5cGUiOiJjb2x1bW4iLCJmdW5jIjoiU1VNIiwieUF4aXMiOiJoZWN0YXJlIiwic2NpZW50aWZpY0Rpc3BsYXkiOnRydWUsImNvbG9yIjoiIzAyNzlCMSJ9XSwieEF4aXMiOiJuZWlnaGJvdXJob29kbmFtZSIsIm1heHBvaW50cyI6NTAsInNvcnQiOiIiLCJzZXJpZXNCcmVha2Rvd25UaW1lc2NhbGUiOiIiLCJjb25maWciOnsiZGF0YXNldCI6InBhcmtzIiwib3B0aW9ucyI6e319fV0sInRpbWVzY2FsZSI6IiIsImRpc3BsYXlMZWdlbmQiOnRydWUsImFsaWduTW9udGgiOnRydWV9&location=11,49.21234,-123.33475

Park facility:

https://opendata.vancouver.ca/explore/dataset/parks-facilities/table/?disjunctive.facilitytype&dataChart=eyJxdWVyaWVzIjpbeyJjb25maWciOnsiZGF0YXNldCI6InBhcmtzLWZhY2lsaXRpZXMiLCJvcHRpb25zIjp7ImRpc2p1bmN0aXZlLmZhY2lsaXR5dHlwZSI6dHJ1ZX19LCJjaGFydHMiOlt7ImFsaWduTW9udGgiOnRydWUsInR5cGUiOiJjb2x1bW4iLCJmdW5jIjoiQVZHIiwieUF4aXMiOiJwYXJraWQiLCJzY2llbnRpZmljRGlzcGxheSI6dHJ1ZSwiY29sb3IiOiIjMDI3OUIxIn1dLCJ4QXhpcyI6ImZhY2lsaXR5dHlwZSIsIm1heHBvaW50cyI6NTAsInNvcnQiOiIifV0sInRpbWVzY2FsZSI6IiIsImRpc3BsYXlMZWdlbmQiOnRydWUsImFsaWduTW9udGgiOnRydWV9