# NF06A

Rapport de projet – « Le Marcel Manager »

Mathurin Lemoine

Printemps

## Table of contents

## Introduction

The goal of this project is to write a program using python and C to build a management system for a bike rental project. There will be multiple functions to simulate a real user impact on the system such as renting, docking a new bike, and repairing the ones that needs it. I chose to do this program on my own because I barely knew anyone in my TD group and wanted to rely on myself only so I have a global view on what I'm doing and what I need to do.

Here is the project's description:

"Among all its mobility services, Troyes Champagne Metropole offers Le Marcel, a self-service electric bicycle rental service, available 7 days a week, 24 hours a day."

The city of Troyes introduced a new electrical bikes rental service called "Le marcel ", with docking stations all over the city. People can rent a bike and return it to any of these stations.

Allowing such flexibility comes at a cost, the city needs to manage these stations and bikes to ensure a good user experience. This includes: bike maintenance, station maintenance, stations load balancing... The city is asking you to build a management system to help them improve their service

## Description of the algorithms

This part will be divided in 7 parts, corresponding to each function and the class definition implemented in the project.

Here is the estimated time spent for each part:

| **Functions // definitions** | **Estimated time** |
|---|---|
| Class definition | 45 mins |
| Function 1: dock_bike() | 1h30 |
| Function 2: display_stations() | 1h |
| Function 3: rent_bike() | 2h |
| Function 4: summary() | 1h |
| Function 5: maintenance_defective_bikes() | 4-5h |
| Function 6: user_panel() | 45 mins |
| Global work | About 13h |

### Class definition:

In the project, we had to have the database of the bikes and the stations in a file. I used the csv (comma separated values) format as it was easier to export / import. For each language, I had to create a class for the bikes and their attributes (UID, battery percentage, number of days since maintenance, number of rents) and do the same for stations and their attributes (UID, location, name, bikes docked to the station, number of rents and returns)

I started by creating the class in python with the built-in class system and made it so it defines itself using the header of my csv file.

For the C part, it was a bit more difficult as there is no built-in class system. I had to create a new data structure to store the needed information for each station.

## Function 1: dock_bike()

This is the first function we were asked to write. It's used to add a new bike to the system, I considered the bike was bought so it has full battery, no rents and no days as it's new. On the first version, the new bike was just added to the bikes and stations class and not to the data files, so each time the program ended, the bike I added last time was not considered.

I wrote the function again to modify the csv files and then when the modifications are pushed, I update the classes

## Function 2: display_stations()

The goal of this function is to display to the user all the stations and the bikes docked to each of them in a descending order according to their battery level.

For each station, if there is one bike, it'll print the bike, else, it will create a new list with the bikes sorted correctly.

Then it will print a line for each station with the UID, the name and the sorted bike list.

## Function 3: rent_bike()

In this part, the user will rent a specific bike for x minutes and will return it to a specific station

First, we have to check if the bike asked has enough battery to cover the time of rent, if not, the user will see a message saying the maximum time of rent for the bike.

If the bike has enough battery, we have to find the station he is attached to, remove it from there and add it to the station he was returned (also have to think about increasing the station's information: number of rents and returns)

Then, because we modified the classes, we have to update the csv data files to fit the changes using pandas

## Function 4: summary()

This will return a summary of the whole system:

- List of bikes according to: number of days in use, number of times rented
- List of stations according to: number of rents, number of returns
- Number of bikes and stations, average battery level of the system

To do the lists, I used the built-in sorting system to create a sorted copy of the stations or bikes depending on the attribute I used.

I then simply printed it in the terminal

## Function 5: maintenance_defective_bikes()

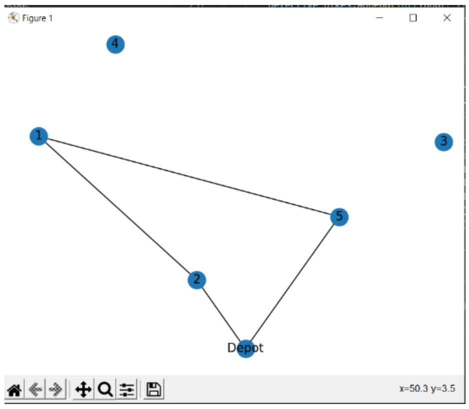
This was the hardest part of the project in my opinion.

In order to ensure the bikes are always in good condition, the city needs to pass by the end of each day by some of the stations in order to get the defective bikes. The goal is to propose the best route that we should take to get all the defective bikes traveling the least distance possible.

This is known as the TSP (traveling salesman problem) and it's a well-known optimization problem. There is no algorithm that gives the exact solution but some deliver an approximated solution in a reasonable time.

The C part will first store each station's UID, and position (other parameters are not needed) in the "class" defined previously. Then the program will look for the path to take to optimize the distance travelled and will store it in an array

Then back to the python part, it will print the resulting path and then by using the network library, a window will pop up with a graphical representation for the previous path and every station in the system.



## Function 6: user_panel()

In this part I simply create a user panel to help navigate through the functions easily and between each execution, the user will have a cooldown of 10 seconds to be able to read the output of the function.

## Problems and how I solved them

During the project, I encountered multiple problems and solved a huge part of them:

- Learning how to use GitHub to keep a backup of the program in case of loss
  **Solution:** Watching tutorials on YouTube and reading forums

- How to self-define the classes
  **Solution:** Reading forums and python documentation

- How to import/export on the csv files
  **Solution:** Used csv and pandas' library and their documentation

- How to use the networkx library
  **Solution:** Read forums, watch videos and read documentation

- Implement the c function in python using CTypes
  **Solution:** Find videos that explains the process, read forums and ask the professor for help to generate the DLL as it was not working

- How to use the doxygen documentation
  **Solution:** Watch videos explaining the process + documentation

## How to use the program

You will find the program files in the same zip as this Report

Extract it and then go to "main.py"

Before using the program, you'll have to install a few libraries by doing this command in the terminal:

```
python3 -m pip install -r requirements.txt
```

It's going to install every package you need to run the python program

When running the program, the user panel will appear with different options corresponding to the functions described before.

To open the doxygen documentation, navigate to the "Doc" folder and open the file called "index.html"

## Conclusion

This project was a real enjoyment and I really loved working on it. It allowed me to apply what we were learning during the classes and see a real-world application to “feel” the code and understand it.

I learned to work on a project with deadlines, deal with failure and use my strengths in the best way

The fact that I used doxygen and GitHub is also very important as I feel it’s useful for every coding project I’ll do in my life!

To do better, I would implement a whole application with tkinter and buttons instead of having to write a number in the terminal. It would allow a better visualization of the results. I would rework probably the C algorithm to optimize it and make it more efficient.

Overall, I’m happy and proud about the work I’ve done alone during the semester.

## Appendix

**Here is the full program with comments:**

```
"""!
@file main.py
@brief Main file of the project.
@author Mathurin Lemoine
@date 2022
@version 1.0
"""
import csv
import ctypes as ct
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from termcolor import colored
import time

def open_dll(name='./testlib.dll'):
    """!
    Load the c library.
    @param name: Name of the DLL file.
    @return: The library.
    """
    return ct.CDLL(name)

c_lib = open_dll()


class Bikes:
    """
    Class to manage the bikes.
    """
    def __init__(self, row, header):
        """!
        Constructor of the class.
        @param row: Row of the CSV file.
        @param header: Header of the CSV file.
        """
        self.__dict__ = dict(zip(header, row))
    def __repr__(self):
        """!
        Representation of the class.
        @return: String.
        """
        return self.UID
```

```python
data = list(csv.reader(open('Bikes.csv'), delimiter=','))
bikes = [Bikes(i, data[0]) for i in data[1:]]


class Stations:
    """
    Class to manage the stations.
    """
    def __init__(self, row, header):
        """!
        Constructor of the class.
        @param row: Row of the CSV file.
        @param header: Header of the CSV file.
        """
        self.__dict__ = dict(zip(header, row))
    def __repr__(self):
        """!
        Representation of the class.
        @return: String.
        """
        return self.UID
data = list(csv.reader(open('Stations.csv'), delimiter=','))
stations = [Stations(i, data[0]) for i in data[1:]]


def dock_bike():
    """
    Dock a new bike at a specific station.
    """
    global bikes
    global stations
    station = int(input('Enter UID of the station you want to dock the new bike: '))
    # UID of the new bike is the next available UID
    bike = str(int(bikes[-1].UID) + 1)
    # If the bike already exists in the system, you can't dock it
    if bike in bikes:
        print('This bike already exists in the system.')
    else:
        df = pd.read_csv('Stations.csv', sep=',')
        df.set_index('UID', inplace=True)
        # if there is no bike in the station, do not put the comma
        if df.at[station, 'Bikes'] == '' or df.at[station, 'Bikes'] == ' ':
            df.at[station, 'Bikes'] = bike
        else:
            df.at[station, 'Bikes'] += f',{bike}'
        df.to_csv('Stations.csv', sep=',')
        # Add the new bike with 100% battery level, 0 days and 0 rents
        df = pd.read_csv('Bikes.csv', sep=',')
        df = df.append({'UID': bike, 'battery_percent': 100, 'Nb_days': 0, 'Nb_rents': 0},
ignore_index=True)
```

```python
        df.to_csv('Bikes.csv', sep=',', index=False, header=True)
        print('Bike added.')
        # Read the modified files and update the class variables
        data = list(csv.reader(open('Stations.csv'), delimiter=','))
        stations = [Stations(i, data[0]) for i in data[1:]]
        data = list(csv.reader(open('Bikes.csv'), delimiter=','))
        bikes = [Bikes(i, data[0]) for i in data[1:]]


#dock_bike()

def display_stations():
    """
    Display all stations and the bikes docked to it in a descending order according to
their battery level.
    """
    print('Stations\nUID\tName\t\tBikes')
    for x in range(len(stations)):
        i = stations[x].Bikes.split(',')
        if len(i) == 0:
            print(f'{stations[x].UID}\t{stations[x].Name}\t\t\t')
        if len(i) == 1:
            print(f"{stations[x].UID}\t {stations[x].Name}\t {stations[x].Bikes}")
        else:
            for y in range(len(i)-1):
                for j in range(0,len(i)-y-1):
                    # Sort the bikes by battery level
                    if int(bikes[int(i[j])-1].battery_percent) < int(bikes[int(i[j+1])-
1].battery_percent):
                        k = i[j+1]
                        i[j+1] = bikes[int(i[j])-1].UID
                        i[j] = bikes[int(k)-1].UID
                    else:
                        pass
            print(f"{stations[x].UID}\t {stations[x].Name}\t {','.join(i)}")

# display_stations()


def rent_bike():
    """
    User will rent a bike from a station, will return it after x minutes to another or the
same station.
    """
    bike = input('Enter bike UID: ')
    bike_uid = int(bike) - 1
    arrival_station = int(input('Enter station UID: '))
    battery_lost = int(input('Enter time of rental: ')) * 2
    bikes[int(bike)-1].battery_percent = int(bikes[int(bike)-1].battery_percent) -
battery_lost
    if bikes[int(bike)-1].battery_percent < 0:
        print(f"You can't rent this bike for {battery_lost//2} minutes, the maximum time
is {bikes[int(bike)-1].battery_percent//2} minutes.")
        bikes[int(bike)-1].battery_percent = int(bikes[int(bike)-1].battery_percent) +
battery_lost
    else :
        bikes[int(bike)-1].Nb_rents = int(bikes[int(bike)-1].Nb_rents) + 1
        for i in range(0,len(stations)):
            # If the bike is docked to the station, remove it from the station
            if bike in stations[i].Bikes:
                stations[i].Bikes = stations[i].Bikes.replace(bike, ' ')
                stations[i].Bikes = stations[i].Bikes.replace(', ,', ',')
                stations[i].Bikes = stations[i].Bikes.replace(', ', '')
                stations[i].Bikes = stations[i].Bikes.replace(' ,', '')
                stations[i].Nb_rents = int(stations[i].Nb_rents) + 1
                departure_station_UID = int(stations[i].UID)
                departure_station = departure_station_UID - 1
            # Add the bike to the returning station
            if int(stations[i].UID) == int(arrival_station):
                stations[i].Bikes += f',{bike}'
                stations[i].Nb_returns = int(stations[i].Nb_returns) + 1
                print(f"The bike {bike} is now docked to the station {stations[i].UID}.")
        # Update the CSV files
        df = pd.read_csv('Stations.csv', sep=',')
        df.set_index('UID', inplace=True)
        if df.at[departure_station, 'Bikes'] == '' or df.at[departure_station, 'Bikes'] ==
' ':
            df.at[departure_station, 'Bikes'] = bike
        else:
            df.at[arrival_station, 'Bikes'] += f',{bike}'
        df.at[arrival_station, 'Nb_returns'] += 1
        df.at[departure_station_UID, 'Bikes'] = stations[departure_station].Bikes
        df.to_csv('Stations.csv', sep=',')

        df = pd.read_csv('Bikes.csv', sep=',')
        df.set_index('UID', inplace=True)
        df.at[bike_uid, 'battery_percent'] = int(bikes[int(bike)-1].battery_percent)
        df.at[bike_uid, 'Nb_days'] = int(bikes[int(bike)-1].Nb_days)
        df.at[bike_uid, 'Nb_rents'] = int(bikes[int(bike)-1].Nb_rents)
        df.to_csv('Bikes.csv', sep=',', header=True)


# rent_bike()
# display_stations()

def summary():
    """
    Summary of the bikes and stations.
    """
    print("There is currently ", str(len(bikes)), " bikes and " , str(len(stations)), "
stations in the system.")
    print("The average battery level is ", round(sum([int(bikes[i].battery_percent) for i
in range(len(bikes))])/len(bikes),2), " %.")

    print(colored('\nBikes sorted by number of days in use', 'white', attrs=['bold',
'underline']))
    print('UID\tName\tNb_returns')
    sorted_bikes_uid1 = sorted(bikes, key=lambda x: int(x.Nb_days), reverse=True)
    for x in range(len(bikes)):
        print(f"{sorted_bikes_uid1[x].UID}\t {sorted_bikes_uid1[x].battery_percent}\t
{sorted_bikes_uid1[x].Nb_days}")
    pass

    print(colored('\nBikes sorted by number of rentals', 'white', attrs=['bold',
'underline']))
    print('UID\tName\tNb_returns')
    sorted_bikes_uid2 = sorted(bikes, key=lambda x: int(x.Nb_rents), reverse=True)
    for x in range(len(bikes)):
        print(f"{sorted_bikes_uid2[x].UID}\t {sorted_bikes_uid2[x].battery_percent}\t
{sorted_bikes_uid2[x].Nb_rents}")
    pass

    print(colored('\nStations sorted by number of rents', 'white', attrs=['bold',
'underline']))
    print('UID\tName\tNb_returns')
    sorted_stations_uid1 = sorted(stations, key=lambda x: int(x.Nb_rents), reverse=True)
    for x in range(len(stations)):
        print(f"{sorted_stations_uid1[x].UID}\t {sorted_stations_uid1[x].Name}\t
{sorted_stations_uid1[x].Nb_rents}")
    pass

    print(colored('\nStations sorted by number of returns', 'white', attrs=['bold',
'underline']))
    print('UID\tName\tNb_returns')
    sorted_stations_uid2 = sorted(stations, key=lambda x: int(x.Nb_returns), reverse=True)
    for x in range(len(stations)):
        print(f"{sorted_stations_uid2[x].UID}\t {sorted_stations_uid2[x].Name}\t
{sorted_stations_uid2[x].Nb_returns}")
    pass

# summary()


def maintenance_defective_bikes():
    """
    Maintenance of all the defective bikes entered by the user.
    """
    defective_bikes = []
    stations_visit = []
    nb_def_bikes = int(input('Enter number of defective bikes: '))
    for i in range(0, nb_def_bikes):
        defective_bikes.append(int(input('Enter UID of the bike: ')))
    # Resetting the nb days of defective bikes
    for i in range(0, nb_def_bikes):
        bikes[defective_bikes[i]-1].Nb_days = 0
    # If the bike is not defective, add a day to the bike
    for i in range(0, len(bikes)):
        if bikes[i].UID not in defective_bikes:
            bikes[i].Nb_days = int(bikes[i].Nb_days) + 1
    # Create the list of stations to visit by maintenance
    for i in range(0,len(stations)):
        station_bikes = stations[i].Bikes.split(',')
        for j in range(0,len(defective_bikes)):
            if str(defective_bikes[j]) in station_bikes:
                stations_visit.append(int(stations[i].UID))
    stations_visit = list(dict.fromkeys(stations_visit))
    array = (ct.c_int * len(stations_visit))(*stations_visit)
    result = (ct.c_int * len(stations_visit))()
    # Execute the tsp algorithm
    c_lib.tsp_with_coords(array, len(stations_visit), result)
    for i in range(0, len(result)):
      print(f"{stations[int(result[i])-1].UID}\t {stations[int(result[i])-1].Name}")

    # Use networkx to display the graph of the result path using coordinates and starting
from (0,0) and going back to (0,0)
    G = nx.Graph()
    G.add_node('Depot', pos=(0,0))
    for i in range(0, len(stations)):
        G.add_node(stations[i].UID, pos=(int(stations[i].x_location),
int(stations[i].y_location)))
    # Add edges to the graph using the result path
    G.add_edge('Depot', stations[int(result[0])-1].UID, weight=0)
    for i in range(0, len(result)):
        if i < len(result)-1:
            G.add_edge(stations[int(result[i])-1].UID, stations[int(result[i+1])-1].UID,
weight=1)
        else:
            G.add_edge(stations[int(result[i])-1].UID, 'Depot', weight=0)
    nx.draw(G, pos=nx.get_node_attributes(G, 'pos'), with_labels=True)
    plt.show()


# maintenance_defective_bikes()

# Create a user panel in the terminal to navigate through the program
def user_panel():
    """
    User panel to navigate through the program.
    """
    print(colored('\nWelcome to the bike rental program!', 'green', attrs=['bold',
'underline']))
    print('\nPlease select an option:\n1. Rent a bike\n2. Dock a bike\n3. Display the
stations\n4. Summary\n5. Execute the maintenance of the defective bikes\n',colored('\r6.
Exit', 'red', attrs=['bold']))
    choice = int(input('\nEnter your choice: '))
    if choice == 1:
        rent_bike()
        print("Showing panel in 10 seconds...")
        time.sleep(10)
        print("Executing the user panel...")
        time.sleep(1)
        user_panel()
    elif choice == 2:
        dock_bike()
        print("Showing panel in 10 seconds...")
        time.sleep(10)
        print("Executing the user panel...")
        time.sleep(1)
        user_panel()
    elif choice == 3:
        display_stations()
        print("Showing panel in 10 seconds...")
        time.sleep(10)
        print("Executing the user panel...")
        time.sleep(1)
        user_panel()
    elif choice == 4:
        summary()
        print("Showing panel in 10 seconds...")
        time.sleep(10)
        print("Executing the user panel...")
        time.sleep(1)
        user_panel()
    elif choice == 5:
        maintenance_defective_bikes()
        print("Showing panel in 10 seconds...")
        time.sleep(10)
        print("Executing the user panel...")
        time.sleep(1)
        user_panel()
    elif choice == 6:
        print(colored('\nThank you for using the bike rental program!', 'red',
attrs=['reverse']))
    else:
        print(colored('\nPlease enter a valid choice!', 'red', attrs=['bold']))
        user_panel()

user_panel()
```

**Now here is the c function part :**

```c
/**
 * @file c-function.c
 * @author Mathurin Lemoine
 * @brief Solving the TSP problem with the C language
 * @version 1.0
 * @date 03/06/2022
 */

#include "c-function.h"
int j = 1;
int i = 0;

/**
 * @brief Structure containing the informations about a station
 * @param UID : Unique ID of the station
 * @param x_locations : x-coordinate of the station
 * @param y_locations : y-coordinate of the station
 */
typedef struct key_value
{
    int UID;
    int x_locations;
    int y_locations;
} dict;

/**
 * @brief Function to calculate the distance between two stations
 * @param[in] a : Station 1
 * @param[in] b : Station 2
 * @return The distance between the two stations
 */
float dist(dict a, dict b) {
    return sqrt(pow(a.x_locations - b.x_locations, 2) + pow(a.y_locations - b.y_locations,
2));
}

/**
 * @brief Function that returns the path of the TSP
 * @param[in] stations_to_visit : List of stations to visit
 * @param[in] size : Size of the list of stations to visit
 * @param[out] order : The order of the stations to visit
 * @return The path of the TSP
 */
void tsp_with_coords(int stations_to_visit[], size_t size, int order[]) {
    // Opening the file containing the informations about the stations and storing them in
the structure previously defined
    FILE *fp;
    char *token;
    fp = fopen("Stations.csv", "r");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
        exit(1);
    }
    char buffer[1024];
    for (char c = getc(fp); c != EOF; c = getc(fp))
    {
        if (c == '\n')
        {
            j++;
        }
    }
    dict values[j + 1];
    int i = 0;
    rewind(fp);
    while (fgets(buffer, 200, fp) != NULL)
    {
        values[i].UID = atoi(strtok(buffer, ","));
        values[i].x_locations = atoi(strtok(NULL, ","));
        values[i].y_locations = atoi(strtok(NULL, ","));
        i++;
    }
    fclose(fp);
    // If the size of the list of stations to visit is 1, the path is the only station
    if (size == 1) {
        order[0] = stations_to_visit[0];
        return;
    }
    // Defining the matrix of distances between the stations
    int cost[size][size];
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            cost[i][j] = dist(values[stations_to_visit[i]], values[stations_to_visit[j]]);
        }
    }
// Finding the minimum cost path
    int min_cost_path[size];
    int min_cost = INT_MAX;
    int min_cost_path_index = 0;
    for (int i = 0; i < size; i++){
        int path[size];
        path[0] = stations_to_visit[i];
        int path_index = 1;
        int path_total = 0;
        // Finding the cost of the path
        for (int j = 0; j < size; j++){
            if (j == i){
                continue;
            }
            path[path_index] = stations_to_visit[j];
            path_index++;
            path_total += cost[i][j];
        }
        // If the cost of the path is lower than the minimum cost, the minimum cost is
updated and the path is stored
        if (path_total < min_cost){
            min_cost = path_total;
            min_cost_path_index = i;
            for (int k = 0; k < size; k++){
                min_cost_path[k] = path[k];
            }
        }
    }
    // Storing the path in the order array
    order[0] = min_cost_path[0];
    for (int i = 1; i < size; i++){
        order[i] = min_cost_path[i];
    }
}
```

**And the c function header:**

```c
/**
 * @file c-function.h
 * @author Mathurin Lemoine
 * @brief C function used for the computation of the solution
 * @version 1.0
 * @date 03/06/2022
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/**
 * @brief Function that returns the path of the TSP
 * @param[in] stations_to_visit : List of stations to visit
 * @param[in] size : Size of the list of stations to visit
 * @param[out] order : The order of the stations to visit
 * @return The path of the TSP
 */
void tsp_with_coords(int stations_to_visit[], size_t size, int order[]);
```