Anna Linden, Chaeah Park, Diep Pham, Mamita Gurung

Prototype Time Traveler Game

Requirement Specification Document

Metropolia University of Applied Sciences

Bachelor’s Degree in Engineering

Information Technology

13 October, 2022

Contents

1 Introduction

The purpose of this Requirement Specification Document is to describe the general idea of the flight simulator game, to present functional and quality requirements for the game and to outline its main idea. The Requirement Specification Document is targeted at the group of developers and teachers of Metropolia University of Applied Sciences. The Document consists of five parts: introduction, vision, functional requirements, quality requirements, and implementation. Chapter 1 of this document is the Introduction to the project and report. Chapter 2 describes the vision by focusing on the main idea of the prototype flight simulator game. Chapter 3 outlines functional requirements by presenting what the user can do with the game. Chapter 4 defines quality requirements including performance requirements and usability requirements. How the project was implemented, coding, GitHub and Database solutions are presented in chapter 5.

According to the Metropolia University of Applied Sciences study plan, the project is divided into two parts: Software 1 and Software 2 courses scheduled for the autumn term in 2022. During Software 1 "Preliminary Project Assignment" is implemented with the goal to build a functional prototype of a flight simulator game. Software 2 deals with "Final project assignment" where the functional prototype is modified and completed. The outcome of the final project is a web-based flight simulator game that uses a map service and an external data source.

2 Vision

In this chapter, the notion of time zones is described as the purpose and the core concept of the game. It is defined what time zone is and how the game supports users to learn about time zones. Also, it explains how the game raises awareness about global warming and CO2 emissions.

2.1 Purpose of the game

The game was created to satisfy the following educational purposes:

1. It helps users learn about time zones and geographical locations of cities.

2. It delivers a message to consider CO2 emissions for sustainability.

When it comes to the first purpose, the game helps users get concrete ideas about the time zone system and geographical locations of cities around the world through visualization.

Understanding time zones is important because it is widely used as a uniform standard for legal, commercial, and social purposes (Time Temperature Inc, 2000)

As Figure 1 depicts on the next page, time zones are geographical regions where the same standard time is used (Merriam Webster, n.d.). Every time zone is defined as an offset from Coordinated Universal Time (UTC) that is from UTC -12:00 to UTC+14:00 (University of Hawaii, n.d.). Time zones are based on the longitude of a place and are defined by Earth's rotation, which completes circling 360 degrees every day (University of Hawaii, n.d.).

Given the importance of usage and the abstract idea of the time zones, it was decided to utilize airport data to help young people learn about time zones.

Figure 1. World Time Zone Map

Last but not least, the game delivers a message to consider sustainability. CO2 emissions are closely linked to climate change. According to the report issued by IPCC, carbon dioxide produced by people need to fall by 45 percent from 2010 levels until 2030 (IPCC, 2015). Besides, global warming has been tackled by United Nations to stop climate change (United Nations, 2015).

In the game, a certain CO2 level is given, and users can play the game within the given conditions. Therefore, users should be cautious about emitting CO2 while playing the game.

2.2 How the game works

2.2.1 The First Stage: Start and read the given information

Figure 2 shows the general graphical user interface of the flight game. In the very beginning, the game asks a user to type a username. Once the user entered the name, three types of information are provided: CO2 budget, the current time of the user's airport, and a different time as a goal.

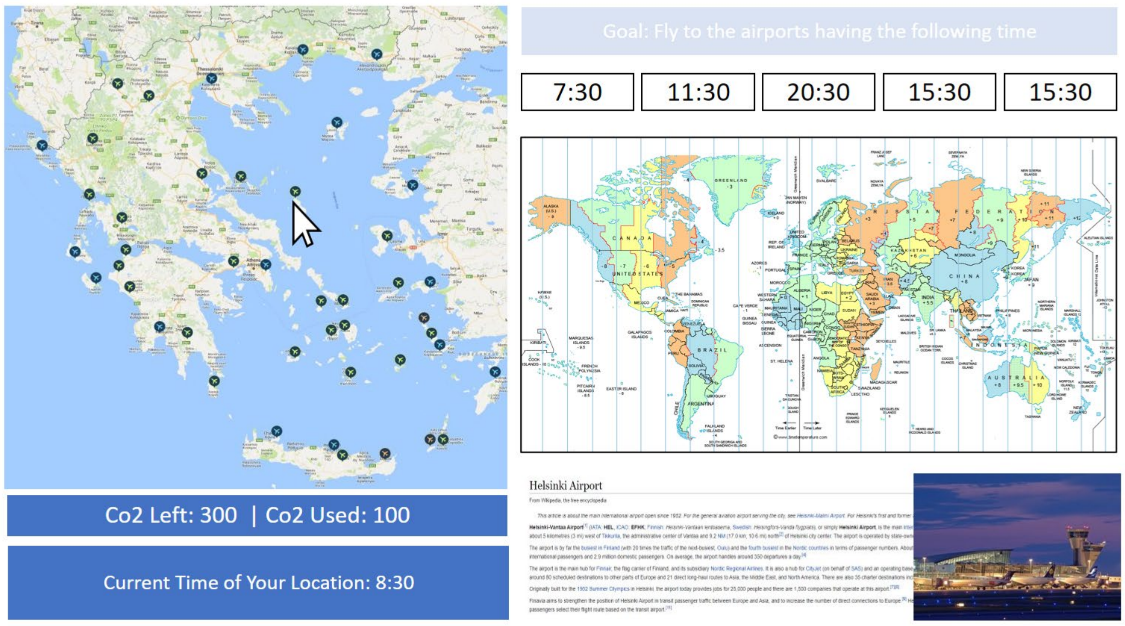

Goal: Fly to the airports having the following time
7:30
11:30
20:30
15:30
15:30
Co2 Left: 300 | Co2 Used: 100
Current Time of Your Location: 8:30
Helsinki Airport

Figure 2. General UI of the game

The user can play the game as long as there is a CO2 budget available. Python randomly picks an airport where the user starts the game from a previously created list of airports. Also, it gets the longitude and latitude of the airport from the database. These two data are used to fetch the current local time with a time API. The game generates a goal within a range of ± 2 hours from the initial time, considering the amount of CO2 emissions.

2.2.2 The Second Stage: Find an airport

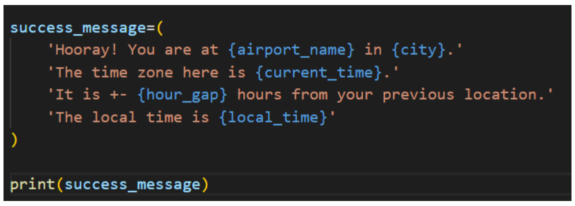
In this stage, the game asks the user to choose an airport that has the same time shown as a goal in the first stage. The user can select one airport from the airport list.

Due to the limit of the text-based interface of this project, the game only uses a list of airports. This approach will be changed by using a map through which the user can select an airport.

2.2.3 The Third Stage: Get a result

Once the user selects the airport, the game calculates the time of the current airport. The longitude and latitude of the current location are used to fetch the current time using a time API.

After the calculation is done, the game displays the result. If the user is in the correct time zone, the message is shown as Figure 3.

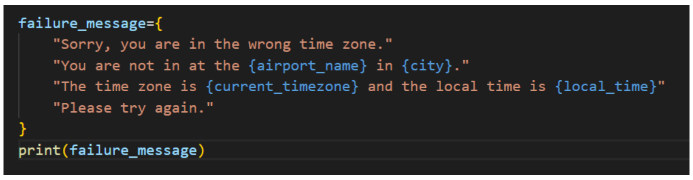


```
success_message=(
    'Hooray! You are at {airport_name} in {city}.'
    'The time zone here is {current_time}.'
    'It is +- {hour_gap} hours from your previous location.'
    'The local time is {local_time}'
)

print(success_message)
```

Figure 3. Example of the success message

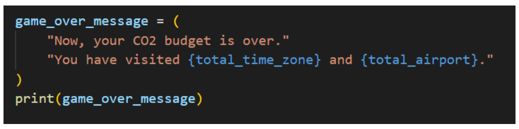
If the user is out of the targeted time zone, the game displays the message demonstrated in Figure 4.



```
failure_message={
    "Sorry, you are in the wrong time zone."
    "You are not in at the {airport_name} in {city}."
    "The time zone is {current_timezone} and the local time is {local_time}"
    "Please try again."
}
print(failure_message)
```

Figure 4. Example of the failure message

In case of no CO2 level left, the user receives the message in Figure 5.



```
game_over_message = (
    "Now, your CO2 budget is over."
    "You have visited {total_time_zone} and {total_airport}."
)
print(game_over_message)
```

Figure 5. Example of the game over message

The message gives feedback, and the user can briefly review the game progress.

3 Functional requirements

In this chapter, all the functional requirements of the flight game are described in terms of user stories to give audiences a concrete view of how the game is played. The requirements are listed in Table 1 below.

Element (role)	What (action)	Why (benefit)
Player	Must be able to fly from one airport to another	To reach an airport that matches the time given as a goal
Airport	Must provide a location and a time zone	So that the player can select an airport as his answer
Time zone	Displays to the player: Initial time	So that the player could see his/her current time zone when he starts the game
	A goal time	Goal time is used by the player as a reference to select a correct airport
CO2	Displays to the player: CO2 budget	Available CO2 budget at the beginning of the game
	The amount of CO2 emissions per flight	The amount of CO2 emissions after flying from one airport to another
	The amount of CO2 budget left	The amount of CO2 left after each flight
Messages	Display to the player information	For the player to get information of: The airport he is located in and the local time
		Whether the correct time zone is reached or not.
		The amount of CO2 budget left.
		The end of the game in case the CO2 budget is over.

Table 1. Functional requirements of the game

The functionality of the flight game will be reviewed and implemented as we develop the game in Software 2 to enhance the user's experience.

4 Quality requirements

This chapter defines the quality requirements of the game designed to ensure that the outcome product fits its educational and entertainment purposes as well as quality satisfies the requirements for the project. According to ISO/IEC 25010:2011 software quality is classified as a structured set of characteristics as follows (ISO/IEC 9126, n.d.):

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Table 2 lists some of the above-mentioned characteristics.

Characteristics	Requirements
Functionality	The current time is fetched from the API.
	The name of the airport and its location shall be fetched through MySQL.
	The latitude and longitude are fetched using geopy module.
	The current time is formatted using DateTime module.
Usability	The rules of the game are easy to understand.
	The game is controlled using the keyboard in the PyCharm run tool window.
	The game has a clear goal and an interesting plot.

Characteristics	Requirements
	The game holds educational value by teaching the concept of time zones.
	The game raises awareness of CO2 emissions and global warming.
Efficiency	The server shall respond within 3 seconds.
	The user must get instant feedback from all actions they perform.
Maintainability	An error message shall be displayed if a problem occurs.
	The game is stable and easy to test.

Table 2. Quality requirements of the game.

5 Implementation

This chapter explains GitHub, database, modular architecture, and code that were implemented throughout the project. Besides, details of reasons and implementation process are demonstrated.

5.1 GitHub

GitHub is an Internet hosting service for code based on the Git version control system (Dabbish, Stuart, Tsay, & Herbsleb, 2012). GitHub was actively used in the project to enhance the collaborative programming process by tracking changes and contributions.

In GitHub, branches play roles as pointers to snapshot code changes. Developers use branches to isolate development work without affecting other branches in the repository. (GitHub Inc., n.d.)

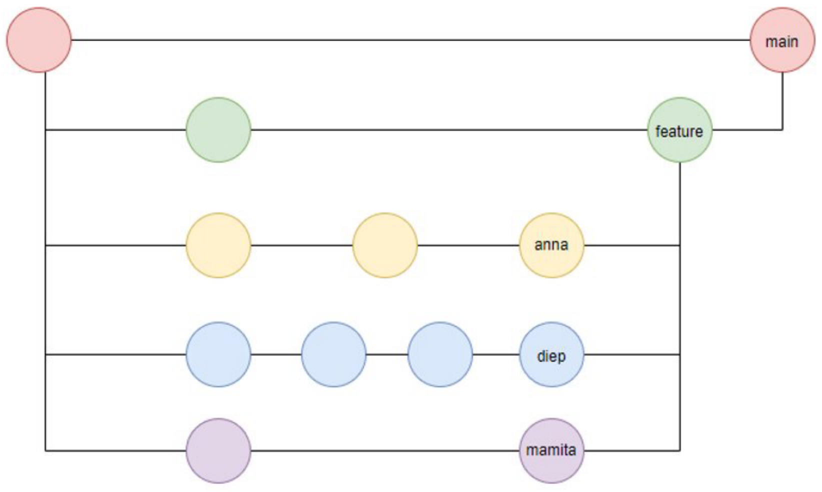


Figure 6. GitHub branch structure applied in the project

Figure 6 above illustrates how branches were structured for this project. Total six branches were created originally, however, five branches were mainly used due to the changes of the members’ responsibilities.

The branches named anna, diep, and mamita were generated as a primary working place for each member. All members pushed codes into the branch where its name matches with their own name. Considering the basic knowledge and experience of GitHub in collaborative projects, the branches were intentionally separated to minimize conflicts occurring during the process when branches are merged.

After creating all logics in the anna, diep, and mamita branches, they were merged into the feature branch. In this process, separated logics in different files were interwoven together and every error was resolved.

After testing the quality of the product in the feature branch, the final product was pushed into the main branch.

5.2 Database

A database system is a collection of programs that runs on a computer and that helps the user to create, read, update, and delete information. (Paredaens, Bra, & Gyssens, 2012)

In this project, a relational database was adopted. The relational database refers to a database based on the relational model of data. (Codd, 1970) There are three important terms regarding the relational database model: Row, Column, and Table.

Rows are data sets representing a single item and columns are names of data such as “latitude_degree” and “longitude_degree” in the Airport table in Figure 7 below. Finally, tables are a set of rows sharing the same attributes and the Airport, Country, and Game in Figure 7 are the examples of tables.

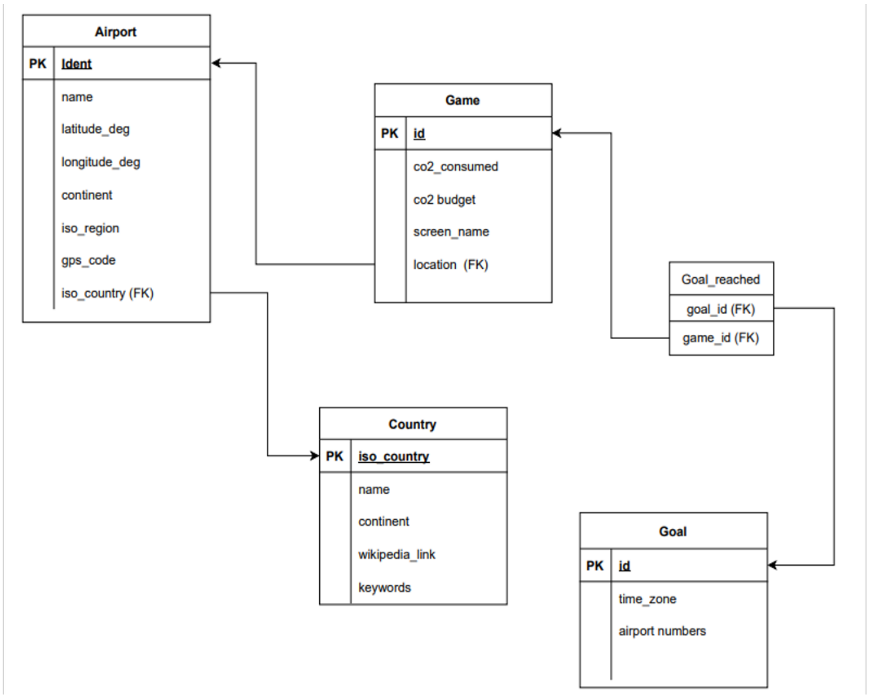


Figure 7. Database structure

In the text-based game, only the Airport and Country table were used. As depicted in Figure 7, the columns of the two tables were modified by omitting unnecessary columns from the original database model in the Relational Database course.

The Game, Goal_reached, and Goal tables will be created in Software 2 and the data within the tables will interact with the graphical user interface.

5.3 Modular architecture and coding solutions.

In this chapter, the structure of the files and coding solutions are explained in more detail.

5.3.1 Modular architecture.

Here's the list of the files with the general explanation of what it contains.

- _main.py - This is the file where all logics from different files are combined together.
- user.py - The user's key information to proceed with the game.
- current_gps.py - Gets the longitude and latitude of an airport. DB is connected in this file.
- current_time.py - Calculates the current time of the initial airport.
- game_goal.py - All logics to generate a game goal (time in a different time zone).
- rand_airport.py - Logics to generate a random airport or a list of airports.
- user_input.py - Functions to take user's inputs.
- co2.py - Calculate the user's co2 budget left.
- message.py - All messages (success, failure, game over) to provide feedback to the user.
- lib.py - Miscellaneous functions used in the project.

5.3.2 Coding solutions.

The game starts in the main file with the if statement shown in Listing 1. The statement gets the user’s name as an input, sets the initial amount of CO2 equal to 600 and initializes the amount of the airports the player has visited. It also calls the function to get the random airport.

```
if user.total_trials == 0:
    user.name = get_user_name()
    user.co2_budget = 600
    user.total_visited_airports = 0
    user.current_airport = get_random_airport()
```

Listing 1. If statement to get the user's name, generate the initial location and initialize the amount of CO2 available.

The function to define the initial location is called get_random_airport and is shown in Listing 2. It is located in the rand_airport.py file and it generates the initial location of the user from a previously created list of airports using the random.choice method.

```
def get_random_airport():
    current_airport = random.choice(starting_point)
    return current_airport
```

Listing 2. Get_ranom_airport function.

After the initial airport is defined, the received information is stored in the user.py file, which is seen in Listing 3. The file also contains the user’s name, the amount of CO2 and the number of visited airports is set to 0.

```
name=""
total_trials = 0
co2_budget = 600
current_airport = ""
new_location = ""
total_visited_airports = 0
```

Listing 3. User.py file.

Listing 4 shows the while loop, which calls the play_game function while the player has some CO2 budget available.

```
while user.co2_budget > 0:
    play_game()
```

Listing 4. While loop initiating the game.

Next, using the get_chosen_airport_gps function seen in Listing 5, the game connects to the local database flight_game using SQL. GPS coordinates of the airport are fetched from the databases and stored in variables called airport_latitude and airport_longitude.

```
def get_chosen_airport_gps(airport_name):
    sql = "SELECT name, latitude_deg, longitude_deg FROM Airport"
    sql += " WHERE name ='" + airport_name + "'"

    cursor = connection.cursor()
    cursor.execute(sql)
    result = cursor.fetchall()

    if cursor.rowcount > 0:
        for row in result:
            latitude =  round(row[1], 2)
            longitude = round(row[2], 2)

            if not user.new_location:
                print(f"This is {row[0]} airport, which has latitude is
{latitude} and longitude is {longitude}. \n")

            global airport_latitude
            global airport_longitude

            airport_latitude = row[1]
            airport_longitude = row[2]

    return [airport_latitude, airport_longitude]
```

Listing 5. get_chosen_airport_gps function.

The function get_airport_time depicted in Listing 6 checks local time through the time API using received latitude and longitude data. The information received from the API is in JSON format, so JSON module is used to work with this data. The datetime.strptime() method creates an object from a received string, which is further formatted to the hour and minute form using strftime() method.

```
def get_airport_time(airport_latitude, airport_longitude):
    response =
requests.get(f'https://timeapi.io/api/Time/current/coordinate?latitude='

f'{airport_latitude}&longitude={airport_longitude}').json()
    init_hour = (response["hour"])
    init_min = (response["minute"])
    init_time = datetime.strptime(f"{init_hour}:{init_min}:00", "%H:%M:%S")
    init_time = init_time.time().strftime("%H:%M")
    print(f"The time of the current airport is {init_time}")
    return {'time': init_time, 'hour': init_hour, 'min': init_min}
```

Listing 6. get_airport_time function.

Using collected information, 5 possible goals for the game are generated. The goals are located within 2 hours difference from the initial place. The reason for such limitation is the amount of CO2, otherwise there would be a possibility that the randomly generated goal time could be too far away and there would not be enough CO2 budget to complete even one trip.

```
def list_of_time_goals(airport_time):
    init_time =
datetime.strptime(f"{airport_time['hour']}:{airport_time['min']}:00",
"%H:%M:%S")

    time1 = init_time + timedelta(hours=2)
    time1 = time1.time().strftime('%H:%M')
    time2 = init_time + timedelta(hours=1)
    time2 = time2.time().strftime('%H:%M')
    time3 = init_time + timedelta(hours=0)
    time3 = time3.time().strftime('%H:%M')
    time4 = init_time - timedelta(hours=1)
    time4 = time4.time().strftime('%H:%M')
    time5 = init_time - timedelta(hours=2)
    time5 = time5.time().strftime('%H:%M')
    list_of_time = [time2,time3,time4,time5]
    return list_of_time

def generate_random_timegoal ():
    time_goal = random.choice (list_of_time_goals())
    print(f"Find in which airort the time is: {time_goal}. ")
    return time_goal
```

Listing 7. list_of_time_goals and generate_random_timegoal functions.

When the time goal is generated, the player is given 5 airports to choose as his further destination. Lists are generated in advance for Software 1 period, so that each airport in the list is located in a different time zone. The process is seen in Listing 8. To simplify the gamer’s experience the user is given numerical options to choose from instead of typing the whole name of the airport. The function returns the name of the new location as a string.

```
def choose_your_destination():
    airport_list = rand_airport.generate_answer_options()
    choice = int(input(f"Choose your next destination.\n"
                       f"If you want to fly to {airport_list[0]}, write 1.\n"
                       f"If you want to fly to {airport_list[1]}, write 2.\n"
                       f"If you want to fly to {airport_list[2]}, write 3.\n"
                       f"If you want to fly to {airport_list[3]}, write 4.\n"
                       f"If you want to fly to {airport_list[4]}, write 5.\n"
                       f"Enter your number: "))
    new_location = ""

    if choice == 1:
        new_location = airport_list[0];
    elif choice == 2:
        new_location = airport_list[1];
    elif choice == 3:
        new_location = airport_list[2];
    elif choice == 4:
        new_location = airport_list[3];
    elif choice == 5:
        new_location = airport_list[4];
    print("\nYou are now in ", new_location, '.')
    return new_location
```

Listing 8. choose_your_destination function

The next step is to calculate the distance between two airports and the amount of CO2 spent during the flight as shown in listing 9. Calsulate_co2 function uses geopy python module to calculate the distance between the airports and then using this number calculates emitted CO2 assuming that the amount of CO2 produced equals 0.115 per each kilometer, which is an average amount of emissions from a Boeing 737-400 per passenger. (Campbell, 2022)

```
from geopy.distance import geodesic as GD
from current_gps import get_chosen_airport_gps

def calculate_co2():
    current_location = get_chosen_airport_gps(user.current_airport)
    new_location = get_chosen_airport_gps(user.new_location)
    chosen_airport = user.new_location

    co2_consumed_per_km = 0.115

    if user.co2_budget > 0:
        user.total_visited_airports += 1

        # CONVERT DECIMAL STR TO NUMBERS.
        current_gps_set = (float(current_location[0]),
float(current_location[1]))
        new_gps_set= (float(new_location[0]), float(new_location[1]))

        # CALCULATE DISTANCE BETWEEN TWO AIRPORTS.
        distance = GD(current_gps_set, new_gps_set).km
        print(f"\nThe distance between {chosen_airport} and
{user.current_airport} is around: ", round(distance, 2), "km")
```

```
        # CALCULATED COMSUMED AND AVAILABLE C02.
        co2_consumed_per_flight = distance * co2_consumed_per_km
        user.co2_budget = user.co2_budget - co2_consumed_per_flight
```

Listing 9. calculate_co2 function

The game interface is shown in Listing 10. After each flight, the player can see a message which states if he reached his time goal or not as well as the amount of CO2 budget he has. After all the CO2 is used the game is over and the player sees how many airports he visited.

```
Enter your name: User
This is Milan Bergamo Airport airport, which has latitude is 45.67 and
longitude is 9.7.
The time of the current airport is 08:04
THE NEW GOAL TIME IS: 08:04

Choose the airport where the local time is 08:04 from the list below.
Choose your next destination.
If you want to fly to Ittoqqortoormiit, write 1.
If you want to fly to Gran Canaria Airport, write 2.
If you want to fly to Warsaw Chopin Airport, write 3.
If you want to fly to Lviv International Airport, write 4.
If you want to fly to Pulkovo Airport, write 5.
Enter your number: 3

You are now in  Warsaw Chopin Airport .
The time of the current airport is 08:04

Hooray! You made it.

The distance between Warsaw Chopin Airport and Milan Bergamo Airport is
around:  1094.4 km
Your flight from Warsaw Chopin Airport to Warsaw Chopin Airport consumed
around: 125.86 kg of CO2.
Current CO2 budget left: 474.14 kg



=============================


This is Warsaw Chopin Airport airport, which has latitude is 52.17 and
longitude is 20.97.

The time of the current airport is 08:04
THE NEW GOAL TIME IS: 09:04
```

Listing 10. Software 1 Time Traveler game interface.

6 References

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communication of thte ACM*, pp. 377-387.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social Coding in GitHub: Transparency and Collaboration. (p. 1278). Seattle: ACM.

Definition of time zone. (n.d.). Retrieved from Merriam Webster: https://www.merriam-webster.com/dictionary/time%20zone

GitHub Inc. (n.d.). *About branches*. Retrieved from GitHub: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

IPCC. (2015). *AR5 Synthesis Report: Climate Change 2014.* Geneva 2: IPCC.

ISO/IEC 9126. (n.d.). Retrieved from https://en.wikipedia.org: https://en.wikipedia.org/wiki/ISO/IEC_9126#Developments

Merriam Webster. (n.d.). *Definition of time zone*. Retrieved from Merriam Webster: https://www.merriam-webster.com/dictionary/time%20zone

Paredaens, J., Bra, P. D., & Gyssens, M. (2012). *The structure of the relational database model.* Springer Science & Business Media.

The natrual environment - Time. (n.d.). Retrieved from University of Hawaii: https://laulima.hawaii.edu/access/content/group/dbd544e4-dcdd-4631-b8ad-3304985e1be2/book/chapter_1/time.htm

Time Temperature Inc. (2000). *World Time Zone Map*. Retrieved from Time Temperature: https://www.timetemperature.com/time-zone-maps/world-time-zone-map.shtml

Time zone. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Time_zone

United Nations. (2015). *Sustainable Development Goals*. Retrieved from UN: https://www.un.org/sustainabledevelopment/sustainable-development-goals/

University of Hawaii. (n.d.). *The natrual environment - Time*. Retrieved from Laulima : https://laulima.hawaii.edu/access/content/group/dbd544e4-dcdd-4631-b8ad-3304985e1be2/book/chapter_1/time.htm