Text

Description automatically generated

**FACULTY OF APPLIED INFORMATION TECHNOLOGY**

**Field of Study: Information Technology**

**Specialty: Computer Science**

Rustem Andassov

No. of student’s record book: 68618

**PROJECT**

**Course: Introduction to programming**

**Topic: Crack the Maze**

**Rzeszow 2023**

Content table

1. Aim of the project
2. Requirements specification

3. Main body

3.1 Design

3.2 Implementation

3.3 Testing results

4. Conclusions

**I. Aim of the project**

The program should help navigate through the jungle by finding one and only path that takes to the exit. The input is a .txt file that contains strings made up of ‘X’s and ‘O’s, where ‘O’ indicates a navigable path and ‘X’ indicates an impassible jungle. The program returns the directions and number of steps to complete in order to escape from the jungle.

The program should:

* Process the input data;
* Find the start;
* Go through ‘O’s (the navigable path);
* Change directions (up, down, right, left);
* Consider possibilities of ending up in a dead-end, thus backtracking to the last divergence of the path;
* Find the exit from the start and remember the path that leads to it;
* Return the directions to escape the jungle.

**II. Requirements specification**

This program requires several specifications in order to function properly. The specifications are following:

1. The input data should be in a **.txt** extension file named **data**;
2. The input file should be in the same directory as the source code;
3. The passable route can be represented only as capital case ‘O’s and impassable jungle only as capital case ‘X’s;
4. There only can be one and only entry point and exit from the jungle (the algorithm was not designed to find the most optimal way to escape the jungle);
5. The entry point (‘O’) can be only in the first column. The exit from the jungle can be only in the last column;
6. The size of the jungle and position of entry and exit points can vary as long as specifications #4 and #5 are followed;
7. There cannot be a separated exit point with no leading route to it from the start. For example:  
     
   XXXXXX  
   XOOXXX  
   OOXXXX  
   XOOOXO  
   XXXXXX

**III. Main body**

**III.I. Design**

Since the task requires a complicated algorithm the program is designed the way that it is divided into 4 functions that are responsible for different functionality to complete the task. The design is as following:

1. **find\_path(jungle, visited, row, col, path)** - is a recursive function that completes the mission of finding the route.
2. **find\_path\_through\_jungle(jungle)** - driver function that calls **‘find\_path()’** function and returns the path or **‘None’** if the path was not found.
3. **show\_directions(jungle)** - supporting function that calls driver function **‘find\_path\_through\_jungle()’** and interprets the indices into directions.
4. **read\_data()** - function to process a text file as an input data.

**III.II. Implementation**

1. **find\_path(jungle, visited, row, col, path)**  
   This is a recursive function that explores different directions (right, left, up, and down) from the current position (**‘row’, ‘col’**) to find the next position in the path. The input **‘visited’** is a 2D list that keeps track of the positions that have already been visited in the current recursive call, so that we don't revisit them. The input **‘path’** is a list of positions that have already been visited in the current recursive call. If the current position is the last column of the grid, the function returns the current **‘path’** plus the current position. Otherwise, the function explores the different directions by making a recursive call to itself for each unvisited position in those directions. If any of those recursive calls returns a non-**’None’** value, it means that a path has been found, so the current function returns that value. If none of the recursive calls returns a non-**’None’** value, it means that no path was found, so the current function returns **‘None’**.
2. **find\_path\_through\_jungle(jungle)**This is the main function that iterates through the rows of the grid and calls the **‘find\_path()’** function for each position in the first column that is represented by 'O'. If **‘find\_path()’** returns a non-**’None’** value, it means that a path has been found, so the function returns that value. If **‘find\_path()’** returns **‘None’** for all positions in the first column, it means that no path was found, so the main function returns **‘None’**.
3. **show\_directions(jungle)**This is supporting function that interprets saved positions into human-readable navigation. It calls **‘find\_path\_through\_jungle()’** and saves it in **‘path’** variable. If there exists an escape route, it saves indices of every step to get to the exit. Then, it calculates how many steps in any direction should be made in order to escape the jungle from the entrance. If there is no path to exit the jungle it prints **“Path has not been found”**.
4. **read\_data()**This function reads every line of **‘data.txt’** file of input data. Appends every String as an element to **‘data’** list and returns it.

**III.III. Testing results**

**First test**

**The input:**

XXXXXXXXXXXXXXXXXXXXXXXXXX

XXXOOOOOOOOOOOOXXXXXXXXXXX

XXXOXXXXXXOXXXOXXXOOOOOOOX

XXXOXXXXXXOXXXOXXXOXXXXXOX

XOXOXOOOXXOXXXOXXXOOOOOXOX

XOXOXOXOXXOXXXOOOXXXXXOXOX

XOXOXOXOXXOXXXXXXXXXXXOXOX

XOXOXOXOXXOOOOOOOOOOOXOXOO

XOXOXOXXXXXXXXXXXOXXOXOXXX

XOXOXOOOOOOOXXXXXOXXOXOXXX

XOXOXXXXXXXOXXXXXOXXOXOXXX

XOOOOOOOOOOOXOOXXOXXOXOXXX

XXXXXXXXXXXOXOXXXOXXOXOXXX

XXXXXXOXXXXOXOXXXOXXOXOXXX

XXOOOOOOOOOOXOXOOOXXOXOXXX

XXOXXXOXXXXXXOXOXOXXOXOOOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXOOOOOXOXOXXOXOXOX

XXXXXXOXXOXXXXXOXOXXOXOXOX

XXXXXXOXXOXOOOXOXOXXXXOXOX

OOOOOOOXXOXXXXXOXOXXXXOXOX

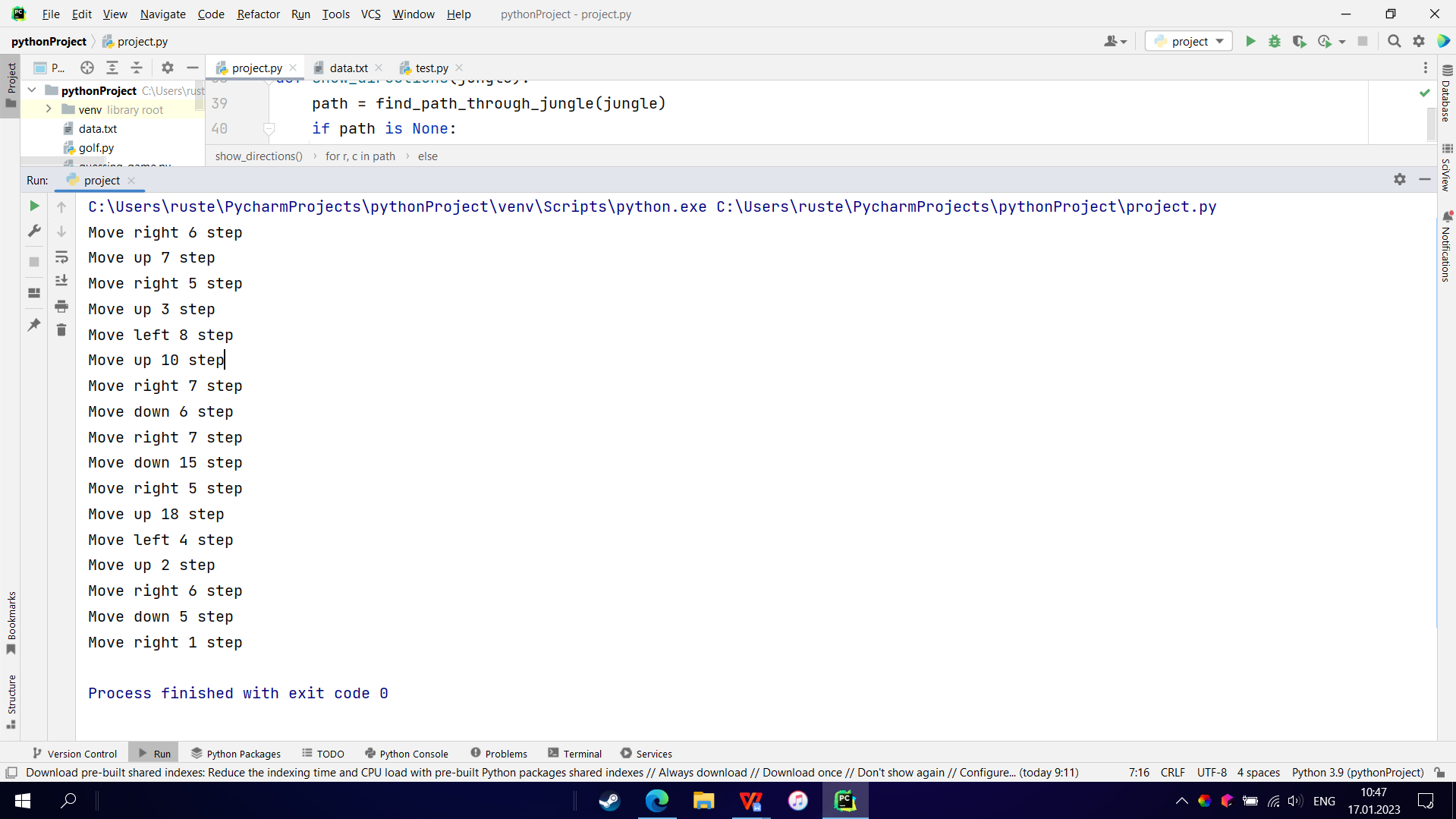
XXXXXXXXXOOOOOOOXOOOOOOXXX

XXXXXXXXXXXXXXXXXXXXXXXXXX

**The expected output:**

The output is expected to show directions following blue marked ‘O’s.

**The output:**



**Second test**

**The input:**

XXXXXXXXXXXXXXXXXXXXXXXXXX

XXXOOOOOOOOOOOOXXXXXXXXXXX

XXXOXXXXXXOXXXOXXXOOOOOOOX

XXXOXXXXXXOXXXOXXXOXXXXXOX

OOXOXOOOXXOXXXOXXXOOOOOXOX

XOXOXOXOXXOXXXOOOXXXXXOXOX

XOXOXOXOXXOXXXXXXXXXXXOXOX

XOXOXOXOXXOOOOOOOOOOOXOXOX

XOXOXOXXXXXXXXXXXOXXOXOXXX

XOXOXOOOOOOOXXXXXOXXOXOXXX

XOXOXXXXXXXOXXXXXOXXOXOXXX

XOOOOOOOOOOOXOOXXOXXOXOXXX

XXXXXXXXXXXOXOXXXOXXOXOXXX

XXXXXXOXXXXOXOXXXOXXOXOXXX

XXOOOOOOOOOOXOXOOOXXOXOXXX

XXOXXXOXXXXXXOXOXOXXOXOOOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXOOOOOXOXOXXOXOXOX

XXXXXXOXXOXXXXXOXOXXOXOXOX

XXXXXXOXXOXOOOXOXOXXXXOXOX

XOOOOOOXXOXXXXXOXOXXXXOXOO

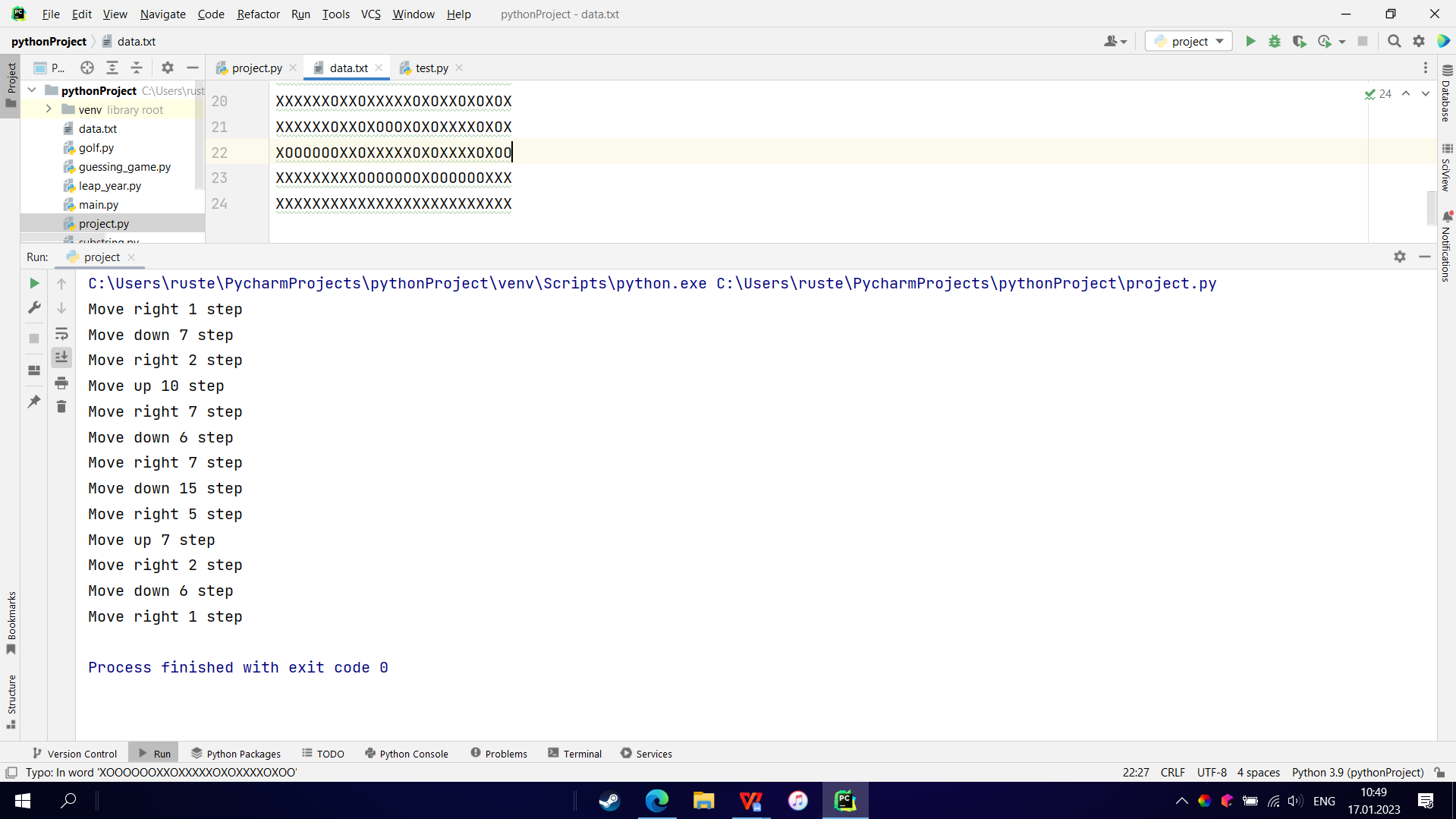
XXXXXXXXXOOOOOOOXOOOOOOXXX

XXXXXXXXXXXXXXXXXXXXXXXXXX

**The expected output:**

In the second test the start and exit coordinates were changed but there is still an escape from the jungle. Expected output should follow blue marked ‘O’s.

**The output:**

**Third test**

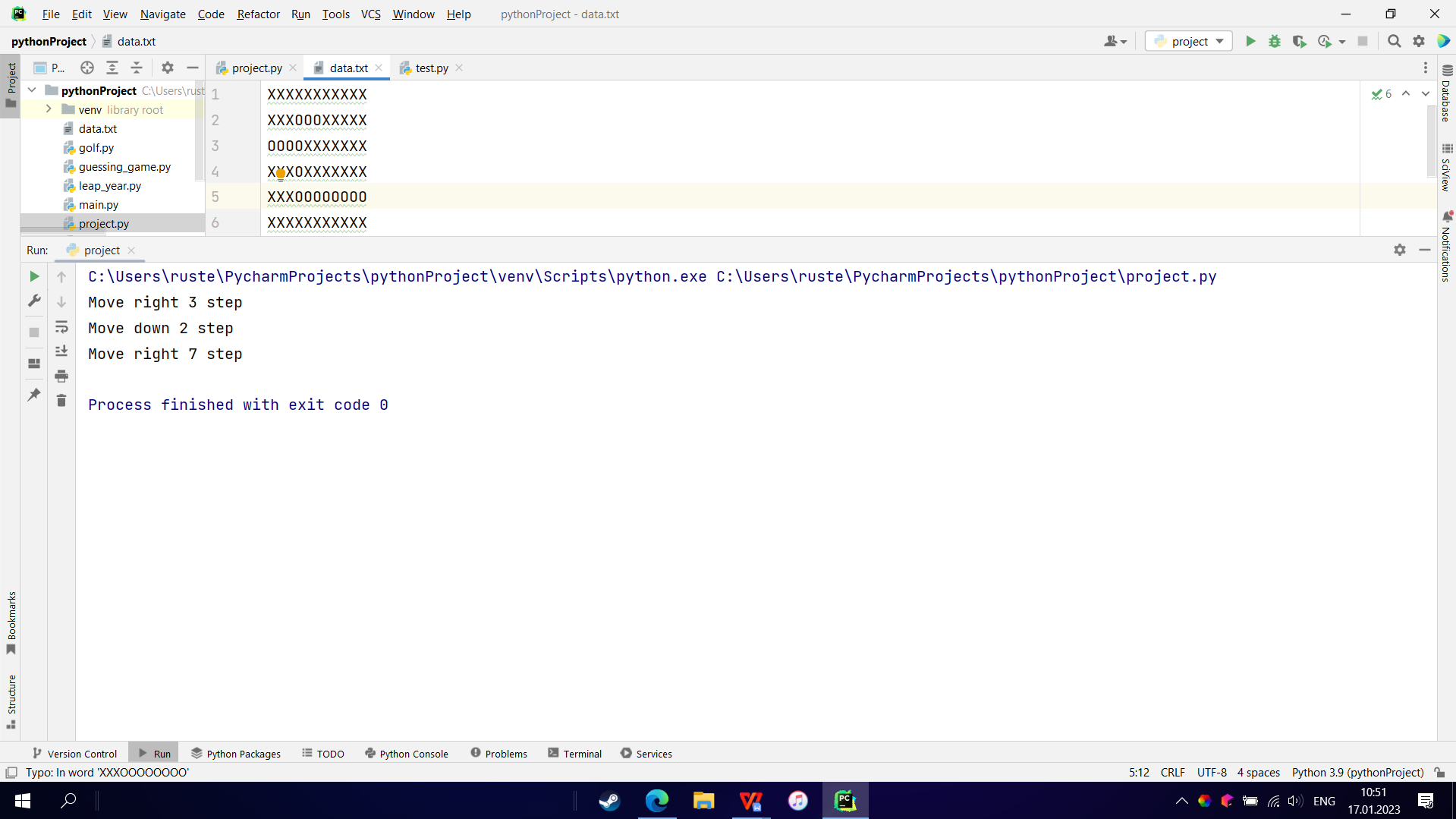
**The input:**

XXXXXXXXXXX  
XXXOOOXXXXX  
OOOOXXXXXXX  
XXXOXXXXXXX  
XXXOOOOOOOO  
XXXXXXXXXXX

**The expected output:**

Even though the input data represents a smaller jungle, it still finds a way out.

**The output:**



**Fourth test**

**The input:**

XXXXXXXXXXXXXXXXXXXXXXXXXX

XXXOOOOOOOOOOOOXXXXXXXXXXX

XXXOXXXXXXOXXXOXXXOOOOOOOX

XXXOXXXXXXOXXXOXXXOXXXXXOX

OOXOXOOOXXOXXXOXXXOOOOOXOX

XOXOXOXOXXOXXXOOOXXXXXOXOX

XOXOXOXOXXOXXXXXXXXXXXOXOX

XOXOXOXOXXOOOOOOOOOOOXOXOX

XOXOXOXXXXXXXXXXXOXXOXOXXX

XOXOXOOOOOOOXXXXXOXXOXOXXX

XOXOXXXXXXXOXXXXXOXXOXOXXX

XOOOOOOOOOOOXOOXXOXXOXOXXX

XXXXXXXXXXXOXOXXXOXXOXOXXX

XXXXXXOXXXXOXOXXXOXXOXOXXX

XXOOOOOOOOOOXOXOOOXXOXOXXX

XXOXXXOXXXXXXOXOXOXXOXOOOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXXXXXOXOXOXXOXOXOX

XXOXXXOXXOOOOOXOXOXXOXOXOX

XXXXXXOXXOXXXXXOXOXXOXOXOX

XXXXXXOXXOXOOOXOXOXXXXOXOX

XOOOOOOXXOXXXXXOXOXXXXOXOX

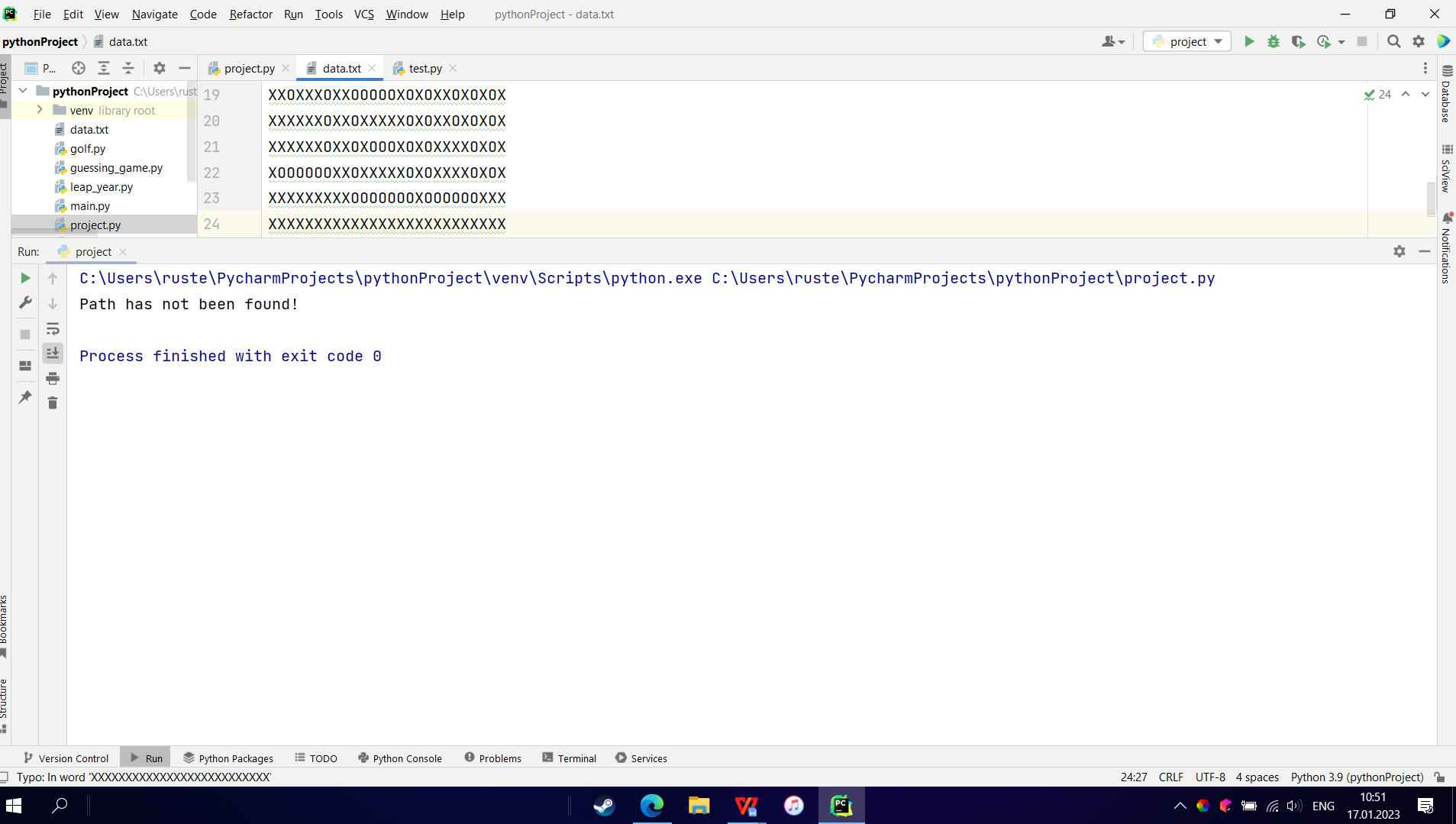
XXXXXXXXXOOOOOOOXOOOOOOXXX

XXXXXXXXXXXXXXXXXXXXXXXXXX

**The expected output:**

In the fourth test program is expected to return “Path has not been found!” as there is no escape from the jungle.

**The output:**



**IV. Conclusions**

There exist several “path finding” algorithms such as: Dijkstra’s algorithm, A\* algorithm and etc. However, we can divide them in two groups: Breadth-First Search(BFS) and Deep-First Search(DFS). Since the task requires us to find “one and only” path, I decided that it is better to use DFS algorithm.

The time complexity of this algorithm is O(*mn*) where *m* and *n* are the number of rows and columns in the input grid, respectively. This is because each position in the grid is visited at most once in the recursive function **‘find\_path()’**, and the function is called at most *m* times in the main function **‘find\_path\_through\_jungle()’**.

The space complexity of this algorithm is also O(*mn*) because the size of the input **‘jungle’** and the **‘visited’** list is O(*mn*) and the maximum depth of the recursive call stack is O(*m*), which means that the maximum size of the **‘path’** list is also O(*m*).

The code uses a recursive *depth-first search* (DFS) algorithm to find the path from the left column of the grid to the right column.

In a DFS algorithm, we explore as far as possible along a path and then backtrack to try other paths. The algorithm uses a recursive approach to explore the different paths, and it terminates the exploration of a path as soon as it reaches the desired goal or determines that the path cannot lead to the goal.

In the implementation, the function **‘find\_path()’** uses a DFS approach to explore the different paths by making recursive calls to itself for each unvisited position in the different directions (right, left, up, and down). The function stops exploring a path and backtracks to the previous position if it reaches a position that is out of bounds, is not represented by 'O', or has already been visited. The function returns a path if it reaches the last column of the grid, or **‘None’** if it cannot find a path.

Honestly speaking, I consider this task as extremely difficult to solve for first-year students. As this kind of task requires students to know, at least, basics of path finding algorithms and moreover the knowledge of applying them. I was able to complete this task only because I have an year experience in programming and have read several books on algorithms’ theory. I would highly recommend you to reconsider this task for first-year students.