ISAD 5004 Final Assignment

Name: Rohan Bansal

Student ID: 21382750

Practical day – Friday (5:00pm – 7:00pm)

Introduction

This assesment is a maze convertion program that uses a made output file, and converts it into box-drawing characters. This project will be using Python to develop the program and perform unittesting. It covers the steps of software enginerring from developing the program with good coding practices, while keeping software engineering practices like modularity in mind. It goes over the design and implementation of Modularity and also Testing. The tsting covers blackbox and whitebox designs and implementation.

Code

Developing the code had several steps to it. The first step was to read the maze file generated by the rubby code and to convert it into an array that would be used by the program. This was achived by the following function:

```
def read_maze(input_filename):
    """This module is responsible for accepting the maze txt output file of the Ruby code
 and converting it into a 2D array.
    The module imports the input_filename variable, which contains the filename for the
 input file. This filename is typed in by the user when running the code.
    The module exports the 2D array that contains the character in the maze txt file."""

    with open(input_filename, 'r') as file:
        return [list(line.strip()) for line in file]
```

The function open thefile by the name/path provided by the user and converts it into an array.

The next function was getting the neighbors of the cell.

```
def get_neighbors(maze, y, x):
    """This module checks the neighbors of the cell and returns the values of the cells in these 4
 positions of the target cell: up, down, left, right.
    Returns the value of the cell in position if it's in range; otherwise, it returns a blank
 whitespace if it's out of range.
    This module imports the maze (2D array containing the Ruby generated symbols) and the for-loop
 counter variables x and y.
    This module also exports the neighbor values which are used by the character_generator module
 (up, down, left, right)."""

    if y > 0:
        up = maze[y-1][x]
    else:
        up = ' '

    if y < len(maze) - 1:
        down = maze[y+1][x]
    else:
        down = ' '

    if x > 0:
        left = maze[y][x-1]
    else:
        left = ' '

    if x < len(maze[y]) - 1:
        right = maze[y][x+1]
    else:
        right = ' '

    return up, down, left, right
```

This was done with the get_neighbors function. It checks the neighbors, up, down, left and right values of the cell (index value provided from the loop) and it returns the neighbors of the taget cell.

```
def character_converter(maze, y, x):
    """This module is responsible for converting the character to box-drawing characters. It goes
 through the neighbors of the cell and based on the characters converts them to box-drawing
 characters.
    This module imports the maze array and the counter variables, x and y.
    This module exports the character. Since this module is called for in a for-loop, it will return
 the character each time it's executed in the loop."""

    up, down, left, right = get_neighbors(maze, y, x)

    if maze[y][x] == '+':
        if up == '|' and down == '|' and right == '-':
            return '\u2523'  # ┣
        elif up == '|' and left == '-' and right == '-':
            return '\u253B'  # ┻
        elif up == '|' and right == '-':
            return '\u2517'  # ┗
        elif up == '|' and left == '-':
            return '\u251B'  # ┛
        elif down == '|' and right == '-':
            return '\u250F'  # ┏
        elif down == '|' and left == '-':
            return '\u2513'  # ┓
        elif up == '|' and down == '|':
            return '\u2503'  # ┃
        elif up == '|' or down == '|':
            return '\u257B'  # ╻
        elif left == '-' and right == '-':
            return '\u2501'  # ━
        elif left == '-' or right == '-':
            return '\u2578'  # ╸
    elif maze[y][x] == '|':
        return '\u2503'  # ┃
    elif maze[y][x] == '-':
        return '\u2501'  # ━'

    return ' '
```

The character_converter function is the main functino of the program that is responsible for converting the charater into the box-drawing character. It check the conditions that are required by the character and converts it accordingly.

Some combinations are excempted as the assignment specified not using some of the characters.

```
def generate_maze(maze):
    """This module is responsible for actually iterating through the array and generating the maze. A
new maze is initialized with empty values,
    these values are then replaced with the converted characters. A nested for-loop is used to
iterate through the two dimensions.
    The module imports the maze array which contains the characters that are going to be converted.
This array is then passed to the character_converter module along with the counter variables that
contain the cell index information.
    The module exports the new_maze array."""
    new_maze = [[''] * len(row) for row in maze]
    for y in range(len(maze)):
        for x in range(len(maze[y])):
            new_maze[y][x] = character_converter(maze, y, x)

    return new_maze
```

The generate_maze function does as it says, generating the maze by replacing the empty values in the newly created array (of the same size as the privded one) and iterating through it by calling the character_converter function.

```
def save_maze(maze, output_filename):
    """This module is responsible for saving the maze as a txt file. It iterates over each
row and converts the list of characters into a string by joining them together without any
separator like (',').
    The concatenated string is then saved as a file with a new line operator after each row
so it's saved in the maze format.
    The module imports the maze array and the name that the user would like to save the new
maze under."""

    with open(output_filename, 'w', encoding='utf-8') as file:
        for row in maze:
            file.write(''.join(row) + '\n')
```

The next step was saving the generated maze, and that was done using the save_maze function like so.

```
def print_maze(maze):
    """This module is responsible for printing the maze to the terminal. The purpose of this module
is to correctly see the maze since opening it with a word processor as a txt file messes up the
format.
    When printed on the terminal, however, it displays the maze correctly.
    The module imports the maze 2D array."""

    for row in maze:
        print(''.join(row))


def main():
    """The module is the main module of the program that brings all the other modules of the program
together.
    It calls upon the other modules as required. It can be considered the master module that runs the
program.
    ** Added a check to make sure user has provided two filenames"""
    if len(sys.argv) != 3:
        print("Please provide two filenames (input)(output)")
        sys.exit(1)
    input_filename = sys.argv[1]
    output_filename = sys.argv[2]

    maze = read_maze(input_filename)
    new_maze = generate_maze(maze)

    print("Original Maze:")
    print_maze(maze)
    print("\n")
    print("New Maze:")
    print_maze(new_maze)

    save_maze(new_maze, output_filename)
    print("\n")
    print(f"New maze saved to {output_filename}")


if __name__ == "__main__":
    main()
```

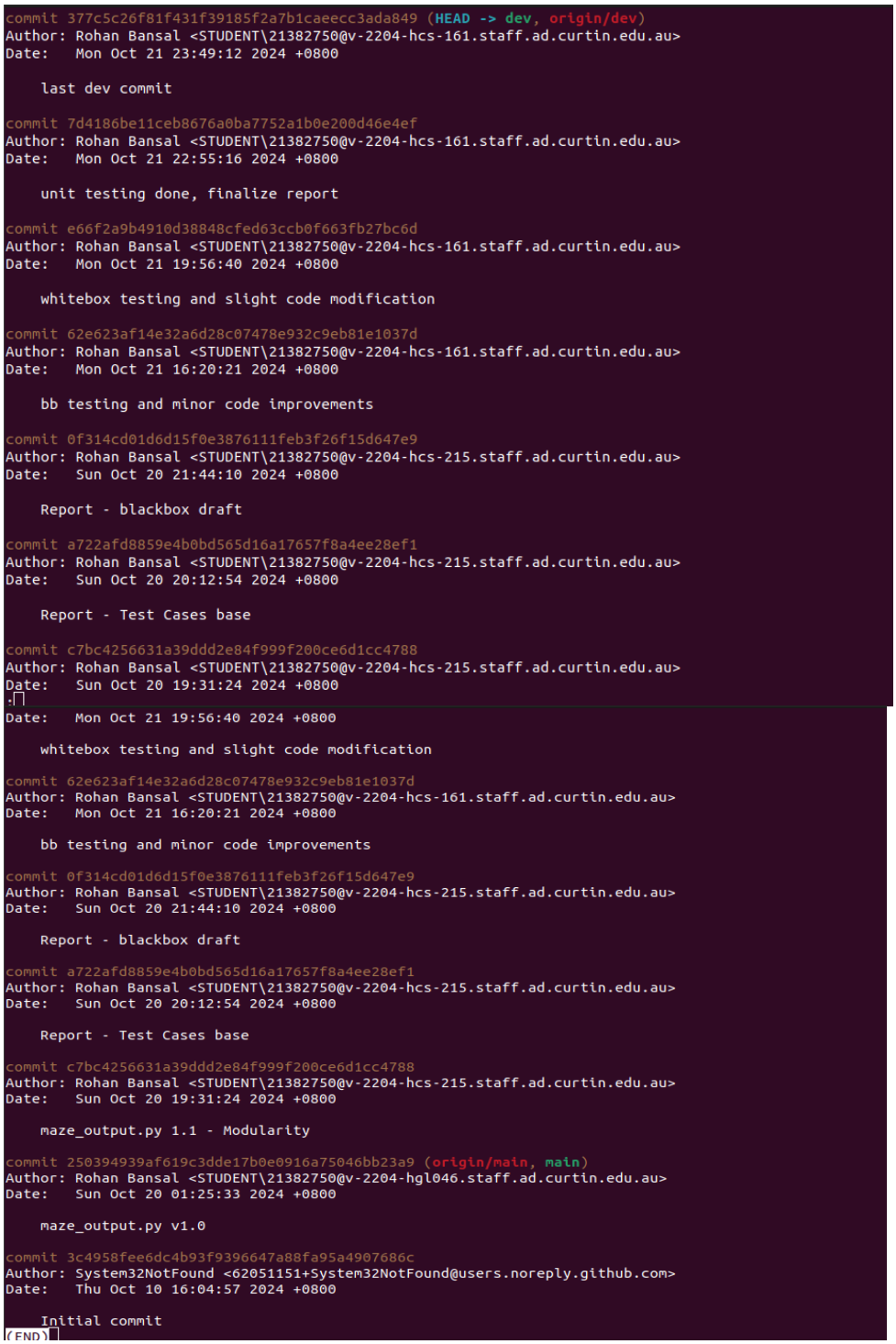
Last two functions are the print_maze and main function.

The print_maze function just print the array by joining the rows,

The main function accpets the input file name and the output file name and uses them to read the maze and pass it to the functions. It also passes the output file name to the save_maze function.

Version Control

Proper use of git was made for this project. Constant commits and pushes were made to ensure safety of data and a branch called dev was also created to ensurea backup of the code in the main branch and continue all the changes in dev branch. Which was later merged into the main branch.



```
    Initial commit

STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ git merge dev
Updating 2503949..377c5c2
Fast-forward
 .~lock.Report.odt#                      |   1 +
 Report.odt                              | Bin 0 -> 2155590 bytes
 __pycache__/maze_output.cpython-310.pyc | Bin 0 -> 5560 bytes
 maze_output.py                          | 120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++--
 maze_output.txt                         |  11 ---------
 new_maze.txt                            |  18 ++++++-------
 new_maze.txxt                           |  11 ++++++++
 unit_testing.py                         | 161 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 8 files changed, 259 insertions(+), 63 deletions(-)
 create mode 100644 .~lock.Report.odt#
 create mode 100644 Report.odt
 create mode 100644 __pycache__/maze_output.cpython-310.pyc
 delete mode 100755 maze_output.txt
 create mode 100755 new_maze.txxt
 create mode 100755 unit_testing.py
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:System32NotFound/Rohan_Bansal21382750_ISE_Repo.git
   2503949..377c5c2  main -> main
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ git log
commit 377c5c26f81f431f39185f2a7b1caeecc3ada849 (HEAD -> main, origin/main, origin/dev, dev)
Author: Rohan Bansal <STUDENT\21382750@v-2204-hcs-161.staff.ad.curtin.edu.au>
Date:   Mon Oct 21 23:49:12 2024 +0800

    last dev commit

commit 7d4186be11ceb8676a0ba7752a1b0e200d46e4ef
Author: Rohan Bansal <STUDENT\21382750@v-2204-hcs-161.staff.ad.curtin.edu.au>
Date:   Mon Oct 21 22:55:16 2024 +0800
```

Commands as such git branch, add, merge, status, log, checkout were used as a part to demonstrate github knowledge.

Modularity Design

Good modularity means low coupling and high cohesion and not using global variables.
The code was already pretty modular since it does not use any global variables, had high cohesion and low coupling.
The code however was in fact refactored, by separating the get_neighbors and character_converter into two separate modules. This was done because the functions have specific functionality and while the character_converter is reliant on get_neighbors, get_neighbors is not reliant on character_converter. This indicates lower coupling and increases cohesion.
Each module was given a clear name indicating it's functionality:

- get_neighbors,
- character_converter,
- generate_maze,
- read_maze,
- save_maze,
- print_maze,
- main

The code uses all variable names that are clear and coherse. The counter values for the for loops are named y and x to make it eaiser to understand 2d array as they represent the index of the array maze[y axis][x axis].
There are total of three functions that call another function in them. While this may seem like high coupling, it is not necesssarily the case. The first function that is depndant on another function is the character_converter, this function like all the other depndant functions is only one way depndant on it, which is still lower coupling. It calls the get_neighbors function as it is required to convert the characters. It is a part of the programming logic that without it the program wouldnt fucntion. It is not possible to remove all cupling from the program. The second function is the genetate_maze function that needs to iterate the character converter. While this may be combined into one function, it would result in lower cohesion as all the functions are working on a specific part. The last function is the main function that requires to call the functions as a manager of the program.

```
def generate_maze(maze):
    """This module is responsible for actually iterating through the a
the maze. A new maze is initialized with empty values,
    these values are then replaced with the converted characters. A ne
to iterate through the two dimensions.
    The module imports the maze array which contains the characters th
converted. This array is then passed to the character_converter module
counter variables that contain the cell index information.
    The module exports the new_maze array."""
    new_maze = [[''] * len(row) for row in maze]
    for y in range(len(maze)):
        for x in range(len(maze[y])):
            new_maze[y][x] = character_converter(maze, y, x)
```

In the main function:

```
    maze = read_maze(input_filename)
    new_maze = generate_maze(maze)

    print("Original Maze:")
    print_maze(maze)
    print("\n")
    print("New Maze:")
    print_maze(new_maze)

    save_maze(new_maze, output_filename)
    print("\n")
    print(f"New maze saved to {output_filename}")
```

In the character_converter function:

```
    return up, down, left, right


def character_converter(maze, y, x):
    """This module is responsible for converting the character to box
 goes through the neighbors of the cell and based on the characters co
 drawing characters.
    This module imports the maze array and the counter variables, x a
    This module exports the character. Since this module is called fo
 will return the character each time it's executed in the loop."""

    up, down, left, right = get_neighbors(maze, y, x)

    if maze[y][x] == '+':
```

Modularity Implementation

(OLD)

```
def cell_character(maze, i, j):
    """Logic to create position checks. Return the value of the cell in position if its in range else it return blank whitespace if its out of range. eg checking for above cell on index [0][y]"""
    if i > 0:
        up = maze[i-1][j]
    else:
        up = ' '

    if i < len(maze) - 1:
        down = maze[i+1][j]
    else:
        down = ' '

    if j > 0:
        left = maze[i][j-1]
    else:
        left = ' '

    if j < len(maze[i]) - 1:
        right = maze[i][j+1]
    else:
        right = ' '

        ''' If satements to decide what symbol gets put down by checking the neighbors'''

    if maze[i][j] == '+':
        if up == '|' and down == '|' and left == '-' and right == '-':
            return '\u2523'  # ┣
        elif up == '|' and down == '|' and left == '-':
            return '\u2503'  # ┃
        elif up == '|' and down == '|' and right == '-':
            return '\u2523'  # ┣
        elif up == '|' and left == '-' and right == '-':
            return '\u253B'  # ┻
        elif down == '|' and left == '-' and right == '-':
            return '\u253B'  # ┻
        elif up == '|' and right == '-':
            return '\u2517'  # ┗
        elif up == '|' and left == '-':
            return '\u251B'  # ┛
        elif down == '|' and right == '-':
            return '\u250F'  # ┏
        elif down == '|' and left == '-':
            return '\u2513'  # ┓
        elif up == '|' or down == '|':
            return '\u2503'  # ┃
        elif left == '-' or right == '-':
            return '\u2501'  # ━
    elif maze[i][j] == '|':
        return '\u2503'  # ┃
    elif maze[i][j] == '-':
        return '\u2501'  # ━

    return ' '
```

(NEW)

```
def get_neighbors(maze, y, x):
    """This module checks the neighbors of the cell and returns the values of the cells in these 4 positions of the target cell: up, down, left, right.
    Returns the value of the cell in position if it's in range; otherwise, it returns a blank whitespace if it's out of range.
    This module imports the maze (2D array containing the Ruby generated symbols) and the for-loop counter variables x and y.
    This module also exports the neighbor values which are used by the character_generator module (up, down, left, right)."""

    if y > 0:
        up = maze[y-1][x]
    else:
        up = ' '

    if y < len(maze) - 1:
        down = maze[y+1][x]
    else:
        down = ' '

    if x > 0:
        left = maze[y][x-1]
    else:
        left = ' '

    if x < len(maze[y]) - 1:
        right = maze[y][x+1]
    else:
        right = ' '

    return up, down, left, right


def character_converter(maze, y, x):
    """This module is responsible for converting the character to box-drawing characters. It goes through the neighbors of the cell and based on the characters converts them to box-drawing characters.
    This module imports the maze array and the counter variables, x and y.
    This module exports the character. Since this module is called for in a for-loop, it will return the character each time it's executed in the loop."""

    up, down, left, right = get_neighbors(maze, y, x)

    if maze[y][x] == '+':
        if up == '|' and down == '|' and right == '-':
            return '\u2523'  # ┣
        elif up == '|' and left == '-' and right == '-':
            return '\u253B'  # ┻
        elif down == '|' and left == '-' and right == '-':
            return '\u253B'  # ┻
        elif up == '|' and right == '-':
            return '\u2517'  # ┗
        elif up == '|' and left == '-':
            return '\u251B'  # ┛
        elif down == '|' and right == '-':
            return '\u250F'  # ┏
        elif down == '|' and left == '-':
            return '\u2513'  # ┓
        elif up == '|' and down == '|':
            return '\u2503'  # ┃
        elif up == '|' or down == '|':
            return '\u257B'
        elif left == '-' and right == '-':
            return '\u2501'  # ━
        elif left == '-' or right == '-':
            return '\u2578'  # ━
    elif maze[y][x] == '|':
        return '\u2503'  # ┃
    elif maze[y][x] == '-':
        return '\u2501'  # ━'

    return ' '
```

Checklist

Item	Checklist Question	Yes/No	Justification
Global Variables			
1	Does the porgram use global variables?	No	Does not use global variables, supports low coupling hence, modularity.
Low Coupling			
2	Does the program pass a large amout of paramters to the modules?	No	Since the parametrs passed are only really the arrray and the counter variables, it displays very low coupling.
3	Do the mocdules depend on control flags to execute the module?	No	Since the modules dont use control flags, it encourages loose coupling.
High Cohesion			
4	Does eah modulec have a specific and defined task?	Yes	Since all the modules have a speceifc task that is well defined, it enourages high cohesion.
5	Does the module have sequential tasks in them?	No (used to)	The get_neighbors and character_converter were intially merged as one function, which discourages high cohesion, this was fixed with a change that made sure they do speicif tasks and do not do multiple tasks in sequene in once function.

Testing Design

Black Box Testing

get_neighbors (maze, y, x) module:
we will assume the maze input as such :

```
[['|', '|', '|'],
['-', '+', '-'],
['|', '|', '|']]
```

Category	Test Data	Expected Result
Top left corner y = 0, x = 0	Y = 0, x = 0	Up = ‘ ‘, down = ‘-’, left = ‘ ‘, right = ‘|’
Bottom right corner y = len(maze) -1, x = len(maze[y]) -1	Y = 2, x=2	Up = ‘-’, down = ‘ ‘, left = ‘|’, right = ‘ ‘
Middle Cell y> 0 AND y < len(maze)-1, X>0 AND X < len(maze[y])-1	Y = 1, x =1	Up = ‘|’, down = ‘|‘, left = ‘-’, right = ‘-‘
Bottom Edge y = len(maze) -1, X>0 AND X < len(maze)-1	Y = 2, x=1	Up = ‘+’, down = ‘ ‘, left = ‘|’, right = ‘|‘
Right Edge y> 0 AND y < len(maze)-1, x = len(maze[y]) -1	Y = 1, x = 2	Up = ‘|’, down = ‘|’, left = ‘+’, right = ‘ ‘

These test cases are supposed to cover different positions in the maze: middle, corners, and edges. This ensures the testing of all possible neighbor combinations.

- Corner pieces have been included of different directions. These pieces are missing 2 directions.
- The middle cell that would return values for all directions has been tested.
- Edge pieces that are missing one direction have also been tested.

This covers all the possibilities of the output the module can produce.

character_converter module:

Category	Test Data	Expected Result
Full Line	Maze = [[' ',' ',' '],['+',' ','+'],[' ',' ',' ']]; y=1, x=0	'\u2503' (┃)
Half Line	Maze = [[' ',' ',' '],['-','+',' '],[' ',' ',' ']]; y=1, x=1	'\u2578' (╸)
Intersection right	Maze = [[' ','|',' '],[' ','+','-'],[' ','|',' ']]; y=1, x=1	'\u2523' (┣)
Intersection up	Maze = [[' ','|',' '],['-','+','-'],[' ',' ',' ']]; y=1, x=1	'\u253B' (┻)
Corner	Maze = [[' ',' ','|'],['+','-','+']]; y=1, x=2	'\u251B' (┛)

These test cases were chosen to cover all the mojor types of box-drawing symbols. These include full line, half line, intersection and corner pieces.

- The full line test case expects a ‘|’ above and below the target cell. However, in case of the target cell being a ‘|’ instead of a ‘+’, it simply replaces it with the box symbol.
- The half line is used when the maze has open walls. This is tested with the second test case.
- Two of the intersections were tested as well. Test case 3 and 4 test when these intersections could take place.
- Lastly the corner case was tested. There are 4 corner characters that are used, top left, top right, bottom left, bottom right. Testing one however, gives the idea of how it works, so the test case tested the bottom right corner piece.

generate_maze module:

Category	Test Data	Expected Result
Maze[y][x]	Maze = [[' ',' ','|'],['+',' ','+'],[' ',' ','|']];	Maze = [[' ',' ',' ┃'],[' ┃',' ',' ┃'],[' ┃',' ',' ┃']];

This module takes in the 2d array that is passed down from the read_maze module. This module would not be passed down an invalid value since if there is something wrong with the provided mzae file, it would be prompted in the red_maze module. So, the test case that was used tests the only test case available for this module, which is providing it with a valid maze and then it calls the character_converter to convert the maze and return the new maze.

read_maze module:

Category	Test Data	Expected Result
Valid filepath\filename	‘maze_input.txt’	Maze converted into array and reutrned. Maze = [["|,' ','|'],['+',' ','+'],['|',' ','|']];
empty field	‘’	Please provide two filenames (input.txt)(output.txt)
Invalid filepath	‘home/maze.txt’ (file does not exist here)	FileNotFoundError: [Errno 2] No such file or directory: 'home/maze_output.txt'

save_maze module:

Category	Test Data	Expected Result
Valid name	‘maze_output.txt’	Files saved under name maze_output.txt in the same directory as python file
empty field	‘’	Please provide two filenames (input.txt)(output.txt)

print_maze module:

Category	Test Data	Expected Result
Valid Array	Maze = [["▌,' ','▌'],['▌',' ','▌'],['▌',' ','▌']];	▌ ▌ ▌ ▌
empty field	‘’	Please provide two filenames (input.txt)(output.txt)

main module:

Category	Test Data	Expected Result
Valid name	‘maze_output.txt’	Files saved and maze generated. Maze = [["▌,' ','▌'],['▌',' ','▌'],['▌',' ','▌']];
empty field	‘’	Please provide two filenames (input.txt)(output.txt)

Whitebox design

The two modules that would benefit from whitebox testing are get_neighbors and character_converter modules.

get_neigbors module:

Path	Test Data	Result
Enter all if, 1st if:(y > 0), 2nd if: (y<len(maze) – 1), 3rd if: (x>0), 4th if: (x<len(maze[y] - 1)	Maze = [[' ','|',' '],['+','-','+'],[' ','|',' ']]; y = 1, x = 1	Up = '|', down = '|', left = '+', right = '+'
Enter 1st else (not y>0) enter all other if	Maze = [['|','|','|'],['+','-','+'],[' ','|',' ']]; y = 0, x = 1	Up = ' '(out of maze) down = '-',left = '|' right = |''
Enter 2nd else (not y < len(maze) – 1) and enter all other if	Maze = [[' ','|',' '],['+','-','+'], ['|','|','|']]; y = 2, x = 1	Up = '-', down = ' '(out of maze), left = '|', right = '|'
Enter 3rd else (not x>0) and enter all other if	Maze = [['|','|',' '],['+','-','+'],['|','|',' ']]; y = 1, x = 0	Up = '|', down = '|', left = ' '(out of maze), right = '-'
Enter 4th else (not x < len(maze[y]) – 1) and enter all other if	Maze = [[' ','|',' '],['+','-','+'],['|',' ','|']]; y = 1, x = 2	Up = ' ', down = '|', left = '-', right = ' ' (out of maze)
Enter 1st and 3rd else and enter 2nd and 4th if	Maze = [[' ','|',' '],['+','-','+'],['|',' ','|']]; y = 0, x = 0	Up = ' ' (out of maze), down = '+', left = ' ' (out of maze), right = '|'
Enter 2nd and 4th else and enter 1st and 2nd if	Maze = [[' ','|',' '],['+','-','+'], ['|','|','|']]; y = 2, x = 2	Up = '+', down = ' ' (out of maze), left = '|', right = ' ' (out of maze)
Enter 1st and 4th else and enter 2nd and 3rd if	Maze = [['|',' ','|'],['+','-','+'],['|',' ','|']]; y = 0, x = 2	Up = ' ' (out of maze), down = '+', left = ' ', right = ' ' (out of maze)
Enter 2nd and 3rd else and enter 1st and 4th if	Maze = [['|',' ','|'],['+','-','+'],['|',' ','|']]; y = 2, x = 0	Up = '+', down = ' ' (out of maze), left = ' ' (out of maze), right = ' '

character_converter module:

Path	Test Data	Result
Enter both if	Maze = [[' ','|',' '],[' ','+','-'],[' ','|',' ']]; y = 1, x = 1	\u2523' (┣)
Enter 1st inner elif	Maze = [[' ','|',' '],['-','+','-'],[' ',' ',' ']]; y = 1, x = 1	'\u253B' (┻)
Enter 2nd inner elif	Maze = [['|',' ',' '],['|',' ','|'],['+','-',' ']]; y = 2, x = 0	'\u2517' (┗)
Enter 3rd inner elif	Maze = [['|',' ',' '],['|',' ','|'],[' ','-','+']]; y = 2, x = 2	\u251B' (┛)
Enter 4th inner elif	Maze = [['+','-',' '],['|',' ',' '],[' ','+','-']]; y = 0, x = 0	'\u250F' (┏)
Enter 5th inner elif	Maze = [[' ','-','+'],['|',' ','|'],['+',' ','+']]; y = 0, x = 2	'\u2513' (┓)
Enter 6th inner elif	Maze = [['|',' ','|'],['+','-','+'],['|',' ','|']]; y = 1, x = 2	'\u2503' (┃)
Enter 7th inner elif	Maze = [['|',' ',' '],['+','-','+'],['|',' ','|']]; y = 1, x = 2	'\u257B' (╻)
Enter 8th inner elif	Maze = [['|',' ',' '],['-','+','-'],['|',' ','|']]; y = 1, x = 1	'\u2501' (━)
Enter 9th inner elif	Maze = [['|',' ',' '],['-','+',' '],['|',' ','|']]; y = 1, x = 1	'\u2578' (╸)
Enter 1st outer elif	Maze = [['|',' ',' '],['+',' ','+'],['|',' ','|']]; y = 0, x = 0	'\u2503' (┃)
Enter 2nd outer elif	Maze = [['|',' ',' '],['+','-','+'],['|',' ','|']]; y = 1, x = 1	'\u2501' (━)

Testing Implementation:

Implementation had few problems when running the unit testing. The expected output was not matching the actual output so it had to be revised. I decided to use an actual maze or the sample_maze as that would allow me to be more flexible and generate each kind of result. Firstly, all the functions from the maze_output.py file were imported. Then a setup file was created and a sample_maze was used. This maze was changed to an actual output maze of the ruby file as it made it easier to test all the cases for the more complex functions.

```
import unittest
import os
import io
import sys
from maze_output import (
    get_neighbors,
    character_converter,
    generate_maze,
    read_maze,
    save_maze,
    print_maze,
    main
)

class TestMazeGenerator(unittest.TestCase):
    def setUp(self):
        self.sample_maze = [['+', '-', '+', '-', '+', '-', '+', '-', '+', '-',
  '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+'], ['|', ' ', ' ', ' ', ' ',
  ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|'],
  ['+', ' ', '+', '-', '+', ' ', '+', ' ', '+', '-', '+', '-', '+', '-', '+', ' ',
  '+', ' ', '+', ' ', ' ', '+'], ['|', ' ', '|', ' ', ' ', ' ', '|', ' ', '|', ' ',
  ' ', ' ', ' ', ' ', ' ', '|', ' ', '|', ' ', '|'], ['+', '-', '+', '-', '+', ' ',
  '+', ' ', '+', '-', '+', ' ', '+', ' ', '+', '-', '+', ' ', '+', '-', '+', ' ', '+'], ['|',
  ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', '|', ' ',
  ' ', ' ', ' ', '|'], ['+', ' ', '+', '-', '+', ' ', '+', ' ', '+', '-', '+', ' ',
  '+', ' ', '+', '-', '+', ' ', '+'], ['|', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|',
  ' ', '|', ' ', '|', ' ', ' ', ' ', ' ', '|'], ['+', '-', '+', '-', '+', '-', '+', '-',
  '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+', '-', '+']]
```

Then the functions test_read_maze, test_save_maze and test_print_maze were created and were tested. For the test_read_maze, a maze file is first created with a small sample maze for testing purposes and then is read by the read_maze function in the main file and the results are compared. For the test_save_module the same approach is taken where a mock maze was created and the file was saved. Then it was checked if that file exists to verify the saving. For testing the print_maze function, test_print_maze function was created, it works by again creating a mock maze then capturing the output. The outputs of the print_maze function is then compared to the captured output to see if they match.

```
def test_read_maze(self):
    # for a valid filepath
    with open('maze_input.txt', 'w') as f:
        f.write("+-+\n| |\n+-+\n")
    result = read_maze('maze_input.txt')
    self.assertEqual(result, [list("+-+"), list("| |"), list("+-+")])
    os.remove('maze_input.txt')

    # For Invalid filepath
    with self.assertRaises(FileNotFoundError):
        read_maze('invalid_maze.txt')

def test_save_maze(self):
    maze = [list("+-+"), list("| |"), list("+-+")]
    save_maze(maze, 'maze_output.txt')
    self.assertTrue(os.path.exists('maze_output.txt'))
    os.remove('maze_output.txt')

def test_print_maze(self):
    maze = [list("+-+"), list("| |"), list("+-+")]
    captured_output = io.StringIO()
    sys.stdout = captured_output
    print_maze(maze)
    sys.stdout = sys.__stdout__
    self.assertEqual(captured_output.getvalue(), "+-+\n| |\n+-+\n")
```

The test_get_neighbors_blackbox does as the name suggests. The sample_maze is used and the expected output is provided to see if the output from the get_neighbors function matches that or now. It compares the the up, down left, right (in that order) cell values match.

```
def test_get_neighbors_blackbox(self):

    # Top left corner
    self.assertEqual(get_neighbors(self.sample_maze, 0, 0), (' ', '|', ' ', '-'))
    # Bottom right corner
    self.assertEqual(get_neighbors(self.sample_maze, 10, 20), ('|', ' ', '-', ' '))
    # Middle Cell
    self.assertEqual(get_neighbors(self.sample_maze, 2, 2), (' ', '|', ' ', '-'))
    # Bottom left corner
    self.assertEqual(get_neighbors(self.sample_maze, 10, 0), ('|', ' ', ' ', '-'))
    # Right Edge
    self.assertEqual(get_neighbors(self.sample_maze, 1, 20), ('+', '+', ' ', ' '))
```

The test_get_neighbors_whitebox method uses a smaller maze for simplicity. It goes through all the if and else paths and compares the output from the get_neighbors function to the given output.

```
    def test_get_neighbors_whitebox(self):
        # Test cases from the white box design
        #decided to use a smaller maze for simplicity for complex unit testing.
        maze = [
            ['+', ' ', '+', ' ', '+'],
            ['|', ' ', '|', ' ', '|'],
            ['+', '-', '+', '-', '+']
        ]
        # Enter all if, 1st if:(y > 0), 2nd if:(y<len(maze) - 1), 3rd if: (x>0), 4th if:
(x<len(maze[y] - 1) -
        self.assertEqual(get_neighbors(maze, 1, 2), ('+', '+', ' ', ' '))
        # Enter 1st else (not y>0) enter all other if -
        self.assertEqual(get_neighbors(maze, 0, 1), (' ', ' ', '+', '+'))
        #Enter 2nd else (not y < len(maze) - 1) and enter all other if -
        self.assertEqual(get_neighbors(maze, 2, 2), ('|', ' ', '-', '-'))
        # Enter 3rd else (not x>0) and enter all other if -
        self.assertEqual(get_neighbors(maze, 1, 0), ('+', '+', ' ', ' '))
        # Enter 4th else (not x < len(maze[y]) - 1) and enter all other if -
        self.assertEqual(get_neighbors(maze, 1, 4), ('+', '+', ' ', ' '))
        # Enter 1st and 3rd else and enter 2nd and 4th if -
        self.assertEqual(get_neighbors(maze, 0, 0), (' ', '|', ' ', ' '))
        # Enter 2nd and 4th else and enter 1st and 2nd if -
        self.assertEqual(get_neighbors(maze, 2, 4), ('|', ' ', '-', ' '))
        # Enter 1st and 4th else and enter 2nd and 3rd if -
        self.assertEqual(get_neighbors(maze, 0, 4), (' ', '|', ' ', ' '))
        # Enter 2nd and 3rd else and enter 1st and 4th if -
        self.assertEqual(get_neighbors(maze, 2, 0), ('|', ' ', ' ', '-'))
```

For the test_character_converter_blackbox function, we test the blackbox categories which are the possible character types by using the sample_maze and providing it with the outputs. The same is done for the whitebox version. There were a lot of errors encountered with these functions. I was initially using a smaller size maze for this and that lead to not enough combinations. So, it was deicded to use an actual maze array and the expected outputs were manually checked for by looking at the sample_maze array.
There were plenty of mismatch errors that had to be correced.

```
def test_character_converter_blackbox(self):

    # Full Line
    self.assertEqual(character_converter(self.sample_maze, 2, 0), '\u2503')
    # Half Line
    self.assertEqual(character_converter(self.sample_maze, 0, 1), '\u2501')
    # Intersection right
    self.assertEqual(character_converter(self.sample_maze, 8, 0), '\u2523')
    # Intersection up
    self.assertEqual(character_converter(self.sample_maze, 10, 16), '\u253B')
    # Corner
    self.assertEqual(character_converter(self.sample_maze, 10, 20), '\u251B')
```

```
def test_character_converter_whitebox(self):

    # Enter both if
    self.assertEqual(character_converter(self.sample_maze, 8, 0), '\u2523')
    # Enter 1st inner elif
    self.assertEqual(character_converter(self.sample_maze, 8, 2), '\u253B')
    # Enter 2nd inner elif
    self.assertEqual(character_converter(self.sample_maze, 10, 0), '\u2517')
    # Enter 3rd inner elif
    self.assertEqual(character_converter(self.sample_maze, 10, 20), '\u251B')
    # Enter 4th inner elif
    self.assertEqual(character_converter(self.sample_maze, 0, 0), '\u250F')
    # Enter 5th inner elif
    self.assertEqual(character_converter(self.sample_maze, 0, 20), '\u2513')
    # Enter 6th inner elif
    self.assertEqual(character_converter(self.sample_maze, 1, 0), '\u2503')
    # Enter 7th inner elif
    self.assertEqual(character_converter(self.sample_maze, 2, 6), '\u257B')
    # Enter 8th inner elif
    self.assertEqual(character_converter(self.sample_maze, 0, 1), '\u2501')
    # Enter 9th inner elif
    self.assertEqual(character_converter(self.sample_maze, 6, 18), '\u2578')
    # Enter 1st outer elif
    self.assertEqual(character_converter(self.sample_maze, 1, 0), '\u2503')
    # Enter 2nd outer elif
    self.assertEqual(character_converter(self.sample_maze, 0, 1), '\u2501')
```

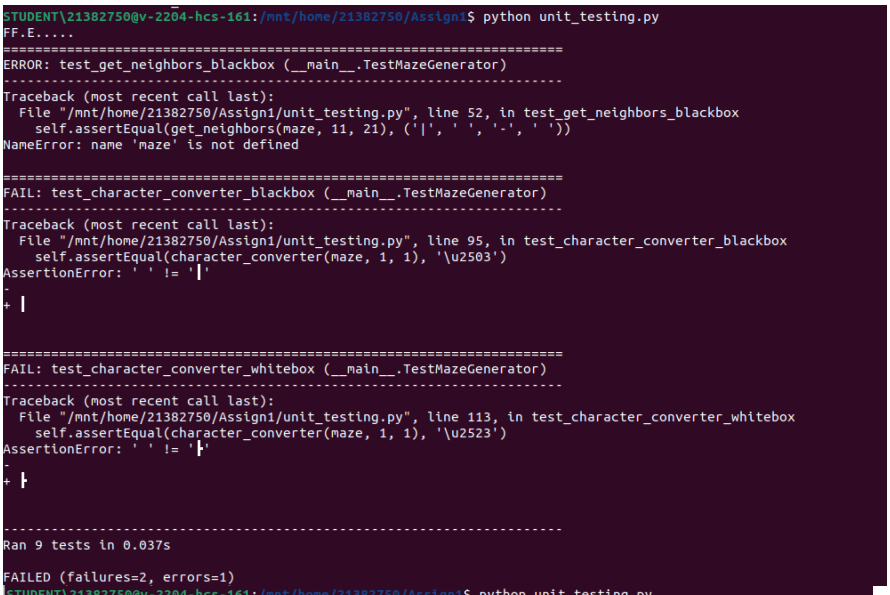
The test_generate_maze function tests if the maze is generating correctly and if all the characters are converting to the box-drawing characters.

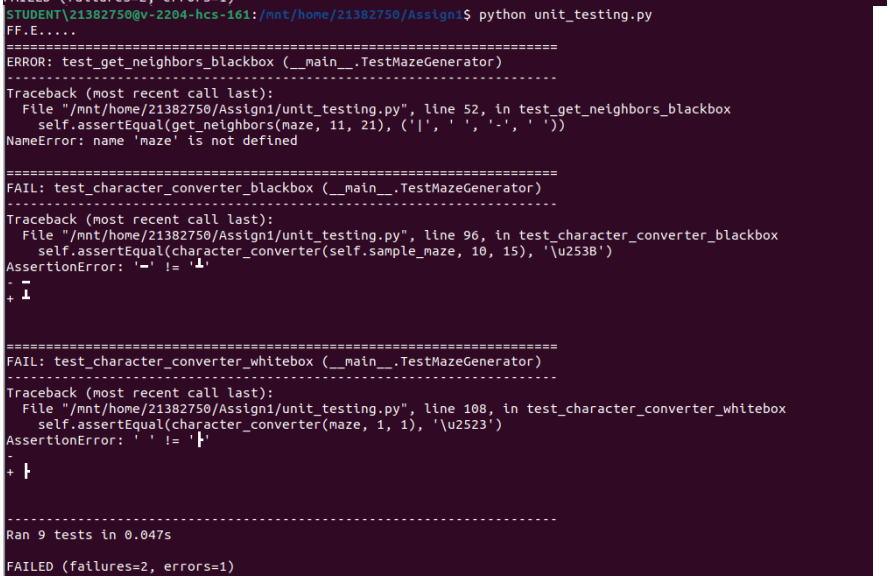
```
def test_generate_maze(self):
    input_maze = [
        ['+', '-', '+'],
        ['|', ' ', '|'],
        ['+', '-', '+']
    ]
    expected_output = [
        ['\u250F', '\u2501', '\u2513'],
        ['\u2503', ' ', '\u2503'],
        ['\u2517', '\u2501', '\u251B']
    ]
    self.assertEqual(generate_maze(input_maze), expected_output)
```

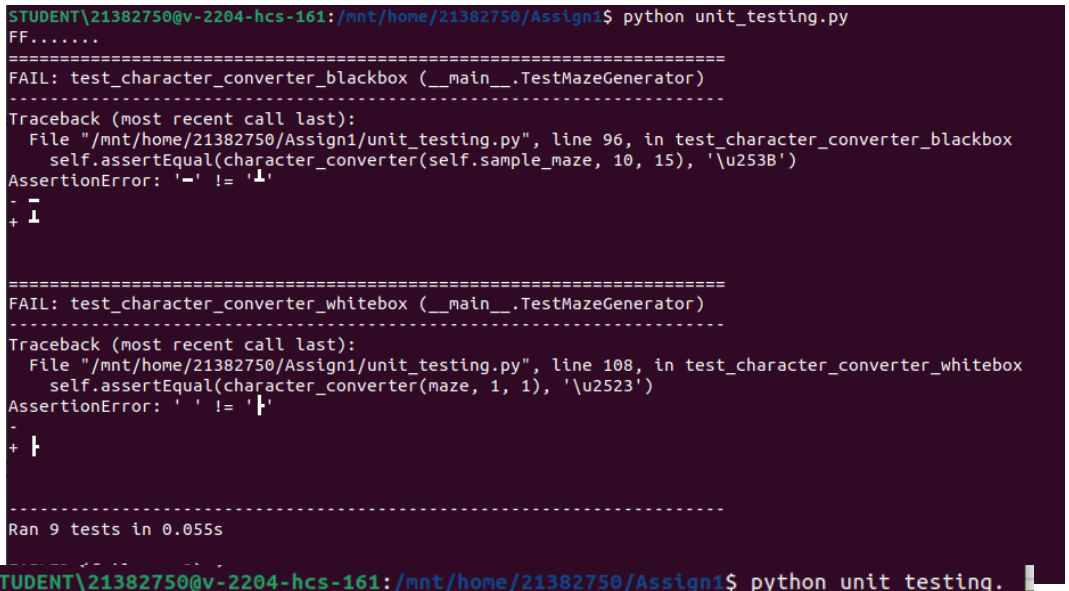
The last testing function is the test_main function
The main function was tested by checking creating a file to act as an input file and then creating an output file and checking if that exists.

```python
def test_main(self):
    # Create a test input file
    with open('test_input.txt', 'w') as f:
        f.write("+-+\n| |\n+-+\n")

    captured_output = io.StringIO()
    sys.stdout = captured_output

    # Run main with test input and output files
    sys.argv = ['maze_generator.py', 'test_input.txt', 'test_output.txt']
    main()

    sys.stdout = sys.__stdout__

    # Check if output file was created
    self.assertTrue(os.path.exists('test_output.txt'))

    # Clean up
    os.remove('test_input.txt')
    os.remove('test_output.txt')
```

Here are a few screenshots demonstrating the progress. (I am only using a few, it took around 15 tries to fix all mistakes.)





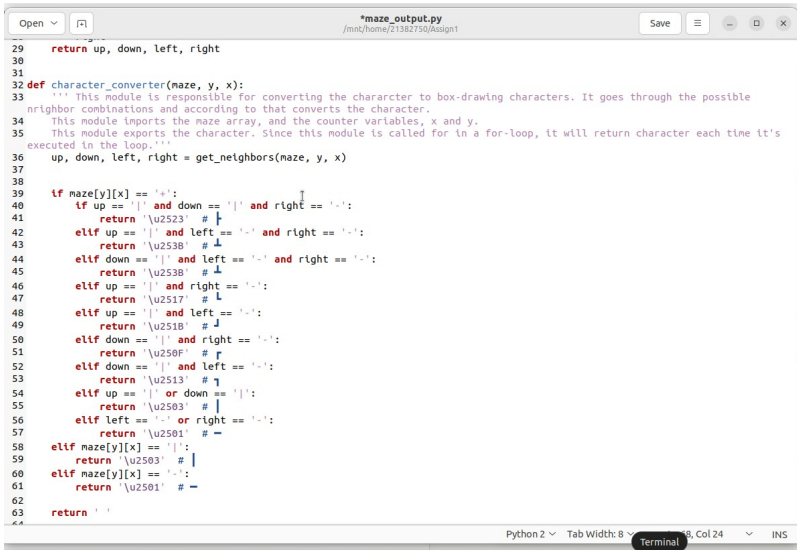

STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ python unit_testing.py
FF.......
FAIL: test_character_converter_blackbox (__main__.TestMazeGenerator)
Traceback (most recent call last):
File "/mnt/home/21382750/Assign1/unit_testing.py", line 96, in test_character_converter_blackbox
self.assertEqual(character_converter(self.sample_maze, 10, 15), '\u253B')
AssertionError: '' != '┻'
FAIL: test_character_converter_whitebox (__main__.TestMazeGenerator)
Traceback (most recent call last):
File "/mnt/home/21382750/Assign1/unit_testing.py", line 108, in test_character_converter_whitebox
self.assertEqual(character_converter(maze, 1, 1), '\u2523')
AssertionError: ' ' != '┣'
Ran 9 tests in 0.055s


STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ python unit_testing.
py
.F.......
FAIL: test_character_converter_whitebox (__main__.TestMazeGenerator)
Traceback (most recent call last):
File "/mnt/home/21382750/Assign1/unit_testing.py", line 121, in test_character
_converter_whitebox
self.assertEqual(character_converter(self.sample_maze, 6, 8), '\u2578')
AssertionError: '┘' != '╸'
Ran 9 tests in 0.045s
FAILED (failures=1)


OK
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$ python unit_testing.
py
....
Ran 9 tests in 0.082s
OK
STUDENT\21382750@v-2204-hcs-161:/mnt/home/21382750/Assign1$

Discussion

There were several hurldes while developing the program.



The functions were different with very little comments. Module descriptions were written inside the code to make sure user can undertstand what each module is for and what it does. Functions were explanded upon and separated to improve modularity.