Procedural Generated Maze Game Application

By Joshua Phillips

For Coder Academy 2023 Term 1 Assignment 3

Features

- Generation of ASCII maze: main feature of application
- Single Maze gaming mode: terminal play of generated maze
- Setting Maze wall colour: customisation of maze colour
- Output Maze to .txt file: save a maze in plain text file for offline play/printing

Features: Generation of ASCII maze

maze_generator.py

```
def generate_maze_ascii(row_maze, column_maze, user_input_difficulty):
```

```
start_point = randint(0, row_maze - 1)
end_point = randint(0, row_maze - 1)

maze_paths[start_point][0] = 1
```

```
# Correct path generation
while True:
    if current_column == (column_maze - 1): #check if right edge
        if current_row == end_point: #on end point, therefore break loop
            break
        elif current_row < end_point:# current position above end point
            next_path_direction = 'down'
        else: # current position below end point
            next_path_direction = 'up'
    else: #not right edge
        next_path_direction = current_moves(current_row, current_column, maze_paths)
    current_path_index += 1
    current_row, current_column = assign_new_path_index(current_row, current_column,
                                                        next_path_direction, current_path_index,
                                                        maze_paths)
```

Correct path generated

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 17, 18]
[2, 0, 6, 7, 0, 0, 0, 15, 16, 19]
[3, 4, 5, 8, 9, 12, 13, 14, 0, 20]
[0, 0, 0, 0, 10, 11, 0, 0, 0, 21]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 22]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 23]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 24]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 25]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 26]
```

Features: Generation of ASCII maze

maze_generator.py

```
# user inputted difficulty used to determine modifer used for fake path quantities
if user_input_difficulty in 'easy':
    fake_modifier = 6
elif user_input_difficulty in 'medium':
    fake_modifier = 5
else: #hard
    fake_modifier = 4
```

```
# total number of fake paths minus the compulsory two fake paths at start and end
fake_path_quantity = randint(column_maze // fake_modifier,
                             (column_maze // fake_modifier) + 2)
```

```
# keep generating random indexes until there are unique amount of indexes
while len(fake_path_set) < fake_path_quantity + 2:
    fake_path_set.add(randint(fake_path_start + 1, fake_path_end - 1))

fake_path_starting_points = list(fake_path_set) #convert set to a list
fake_path_starting_points.sort() # sort fake path starting points
correct_to_fake_connection = [] # contain where correct path and fake pat
```

```
for fake_starting_point in fake_path_starting_points:
    # find where fake starting point is on the maze path index list
    for row_index in range(len(maze_paths)):
        if fake_starting_point in maze_paths[row_index]:
            current_row = row_index
            current_column = maze_paths[row_index].index(fake_starting_point)
            break

    # Fake path generation
    first_fake_step = 1
    while True:
        # check if there is a valid path next to current point (i.e is there a spot left,right,up or
        current_path_index += 1
        next_path_direction = current_moves(current_row, current_column,
                                            maze_paths, 0)
        if "blocked" in next_path_direction :
            break
        current_row, current_column = assign_new_path_index(current_row, current_column,
                                                            next_path_direction, current_path_index,
                                                            maze_paths)
        if first_fake_step: # only want first fake step
            correct_to_fake_connection.append((fake_starting_point, current_path_index))
            first_fake_step = 0
```

Correct path generated

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 17, 18]
[2, 0, 6, 7, 0, 0, 0, 15, 16, 19]
[3, 4, 5, 8, 9, 12, 13, 14, 0, 20]
[0, 0, 0, 0, 10, 11, 0, 0, 0, 21]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 22]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 23]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 24]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 25]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 26]
```

Fake paths generated around correct path

```
[0, 0, 0, 31, 32, 33, 34, 35, 36, 37]
[1, 28, 29, 30, 40, 41, 42, 43, 17, 18]
[2, 27, 6, 7, 39, 0, 0, 15, 16, 19]
[3, 4, 5, 8, 9, 12, 13, 14, 45, 20]
[0, 0, 0, 0, 10, 11, 48, 47, 46, 21]
[0, 0, 0, 0, 0, 50, 49, 58, 57, 22]
[0, 0, 0, 0, 0, 51, 52, 59, 56, 23]
[0, 0, 0, 0, 0, 0, 53, 54, 55, 24]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 25]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 26]
```

Features: Generation of ASCII maze

maze_generator.py

```
# create list to hold ascii maze
maze_ascii = [[" " for _ in range((column_maze * 2) - 1)] for _ in range((row_maze * 2) - 1)]
# evaluate and assigned ascii values to maze
end_point_value = maze_paths[end_point][-1]
for row_index in range(row_maze):
        for column_index in range(column_maze):
            evaluate_directions_and_assign(row_index, column_index,
                                           maze_paths, maze_ascii,
                                           end_point_value, correct_to_fake_connection)

# input starting player position and end point (X)
maze_ascii[start_point * 2][0] = '@'
maze_ascii[end_point * 2][-1] = 'X'
#return maze ascii and row position of start and end point
return maze_ascii, start_point * 2, end_point * 2
```

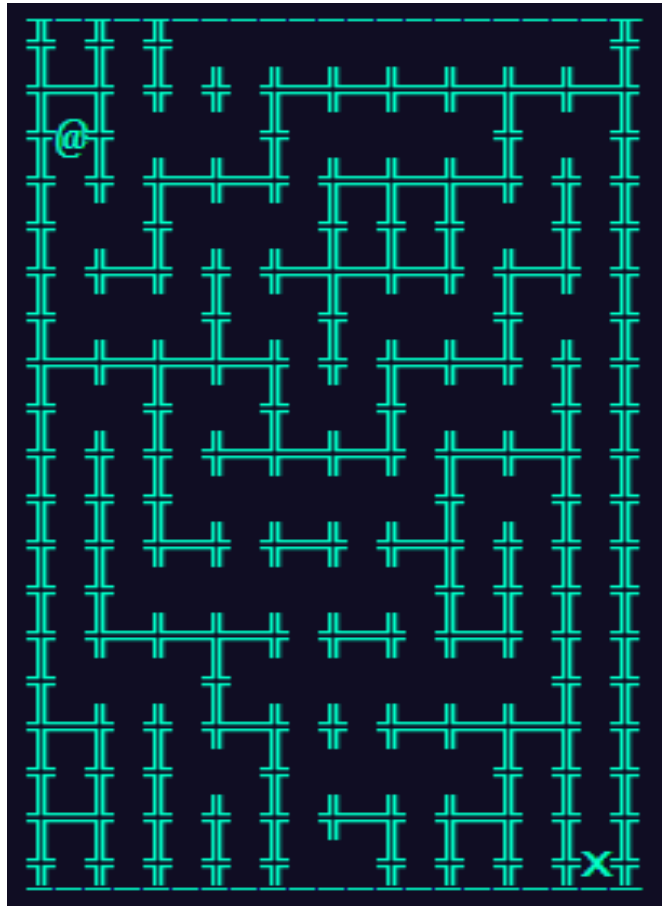
```
def evaluate_directions_and_assign(row_position, column_position, maze_index,
                                   maze_final, end_value, correct_fake_connection):
    maze_value = maze_index[row_position][column_position]

    if column_position <= len(maze_index[0]) - 2: # all columns except for last column
        maze_next_value = maze_index[row_position][column_position + 1]
        maze_final[row_position * 2][(column_position * 2) + 1] = check_direction(maze_value, maze_next_value, 'right',
                                                                                  end_value, correct_fake_connection)

    if row_position <= len(maze_index) - 2: # all rows except for last row
        maze_next_value = maze_index[row_position + 1][column_position]
        maze_final[(row_position * 2) + 1][column_position * 2] = check_direction(maze_value, maze_next_value, 'down',
                                                                                  end_value, correct_fake_connection)
        if column_position <= len(maze_index[0]) - 2: # all except for bottom right corner of maze
                maze_final[(row_position * 2) + 1][(column_position * 2) + 1] = chr(0x256C)
```

```
[0, 0, 0,
[1, 28, 29,
[2, 27, 6,
```

0	M	0	M	0
W	W	C	W	C
1	W	28	C	29
C	W	C	W	W
2	C	27	W	6



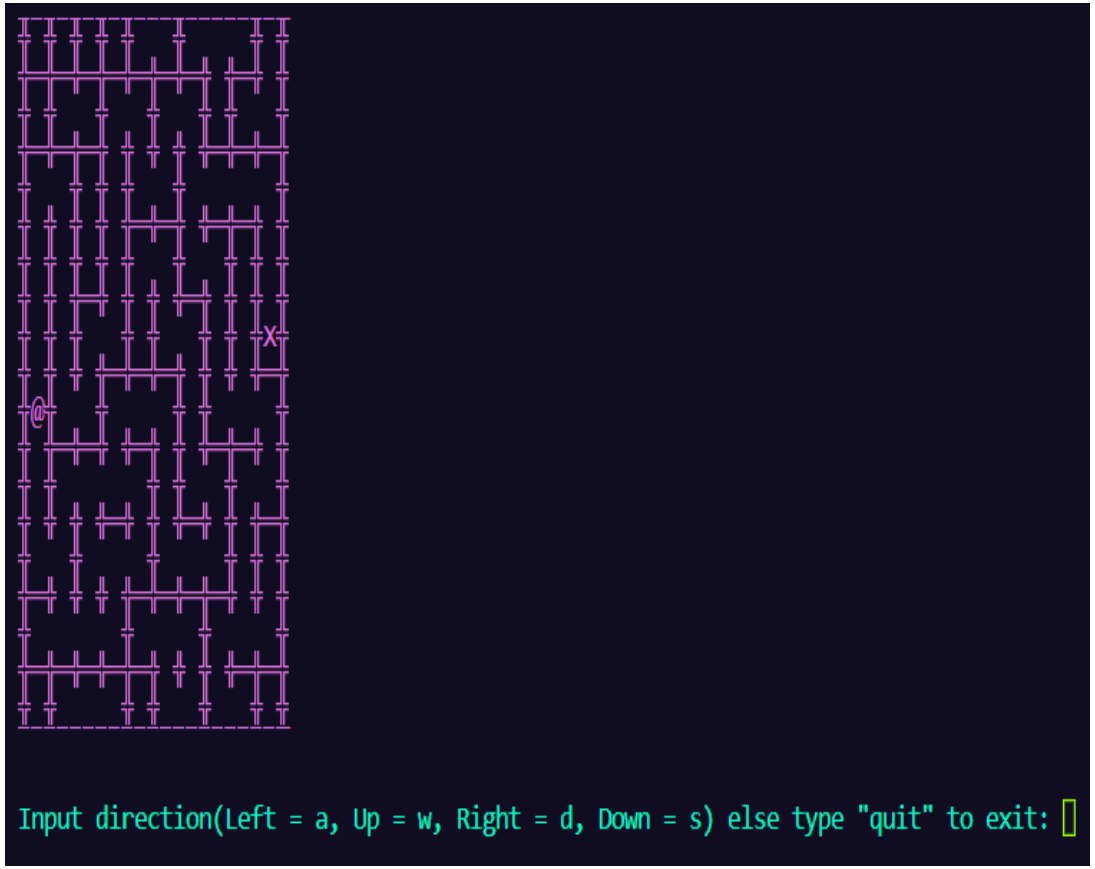
Features: Single Maze gaming mode

maze_modes.py

```python
def single_play_mode():
    maze, player_row, player_column, finish_row, fg_colour_int = new_game()

    while True: #keep goin until player wins game or quits
        if player_column == len(maze[0]) - 1 and player_row == finish_row:
            repeat_game = input("You have won, play again? (yes/no): ")
            if 'yes' == repeat_game:
                maze, player_row, player_column, finish_row, fg_colour_int = new_game()
            elif 'no' == repeat_game:
                raise KeyboardInterrupt
            else:
                print("Invalid input (input: 'yes' or 'no')")
                continue
        player_row, player_column = print_maze_and_move(maze, player_row,
                                                        player_column, fg_colour_int)
```

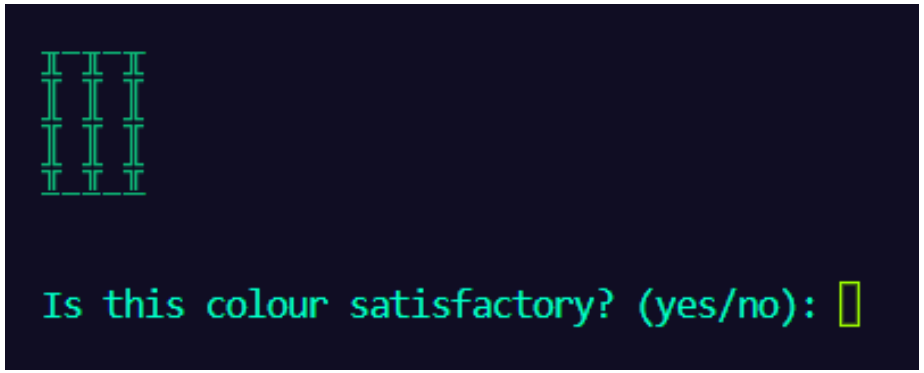
```python
def new_game():
    maze, start_row, end_row = user_input_maze_output()
    start_column = 0
    fg_int = wall_colour_selection()
    return maze, start_row, start_column, end_row, fg_int
```



Features: Setting Maze wall colour

maze_modes.py

```
def wall_colour_selection():
    colour_test_maze = ["_____",
                        chr(0x256c) + " " + chr(0x256c) + " " + chr(0x256c),
                        chr(0x256c) + " " + chr(0x256c) + " " + chr(0x256c),
                        chr(0x256c) + " " + chr(0x256c) + " " + chr(0x256c),
                        chr(0x203e) + chr(0x203e) + chr(0x203e) + chr(0x203e) + chr(0x203e)]
    while True:
        foreground_colour = input("What colour for maze walls? ('red','green','pink','white'): ")
        match foreground_colour:
            case 'red':
                colour_int = 1
            case 'green':
                colour_int = 2
            case 'pink':
                colour_int = 13
            case 'white':
                colour_int = 15
            case _:
                print("Invalid colour")
                continue
        system('cls' if name == 'nt' else 'clear')
        print(f'{fg(colour_int)}')
        for line in colour_test_maze:
            print(line)
        print(f'{attr(0)}')
        while True:
            confirm_colour = input("Is this colour satisfactory? (yes/no): ")
            if "yes" == confirm_colour:
                return colour_int
            elif "no" == confirm_colour:
                break
            else:
                print("Invalid input ('yes' or 'no')")
```



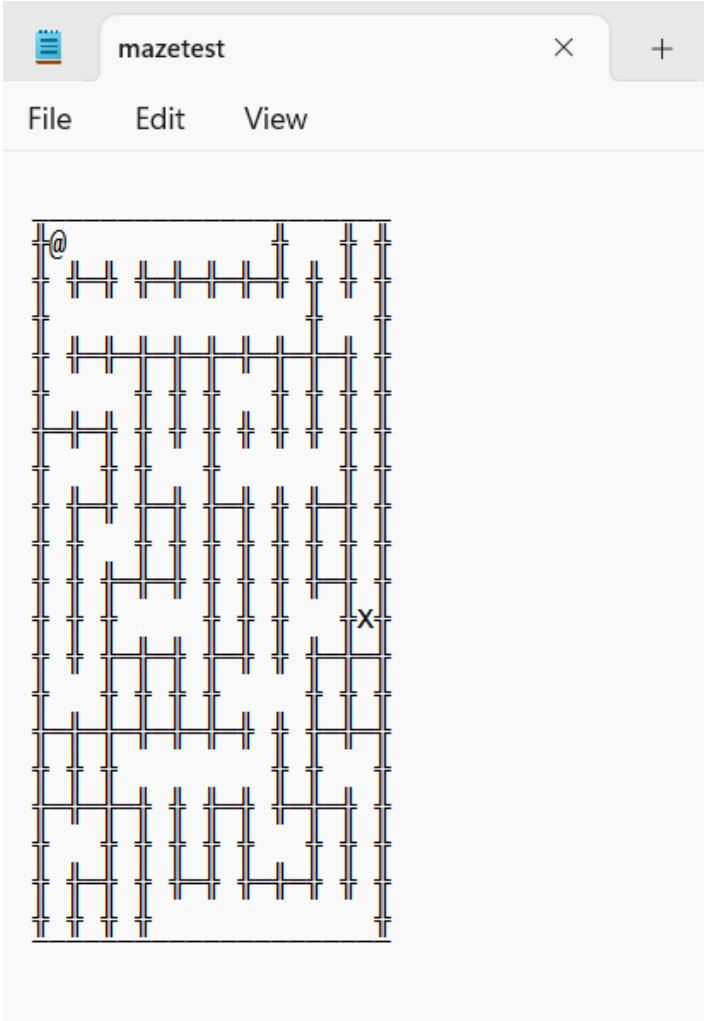
Features: Output Maze to .txt file

maze_modes.py

```python
def text_file_mode():
    # Prompt user for file name and check if valid
    try:
        while True:
            maze_name = input("Enter in name for maze file. No spaces and only alphabetic and numbers (e.g. 'maze1'): ")
            if maze_name.isalnum():
                user_text_file = maze_name + '.txt'
                if path.exists(user_text_file): #file exists
                    while True:
                        confirm_maze_overwrite = input(f'{user_text_file} already exists, do you want to overwrite? (y/n): ')
                        if confirm_maze_overwrite == 'y':
                            raise MazeFileOverwrite
                        elif confirm_maze_overwrite == 'n':
                            break
                        else:
                            print("Invalid input 'y' or 'n' only")
                else:
                    break
            else:
                print("Name can only be alphabetic and numbers, no spaces")
    except MazeFileOverwrite:
        print(f'{user_text_file} will be overwritten with new maze')
    else:
        print(f'New file {user_text_file} will be created')

    # generate maze
    maze, _, _ = user_input_maze_output()
    #convert maze to string list
    mazeString_list = maze_to_stringList(maze)
    # put in file name from user into with open(userfile.txt, 'w')
    with open(user_text_file, 'w') as maze_file:
        for line in mazeString_list:
            maze_file.writelines(line)

    # when finished, raise KeyBoardException to exit program
    print(f'maze saved in {user_text_file}')
    raise KeyboardInterrupt
```



Live demo: How to use application

Review of application project

- Good learning experience. Got better at problem solving and python coding, especially error handling.
- Challenged initially on how to create a maze.
- Would of liked to do more test driven development.
- My Bash script is a bit too simple. Would like to learn more about it.
- Trello board was a big help.