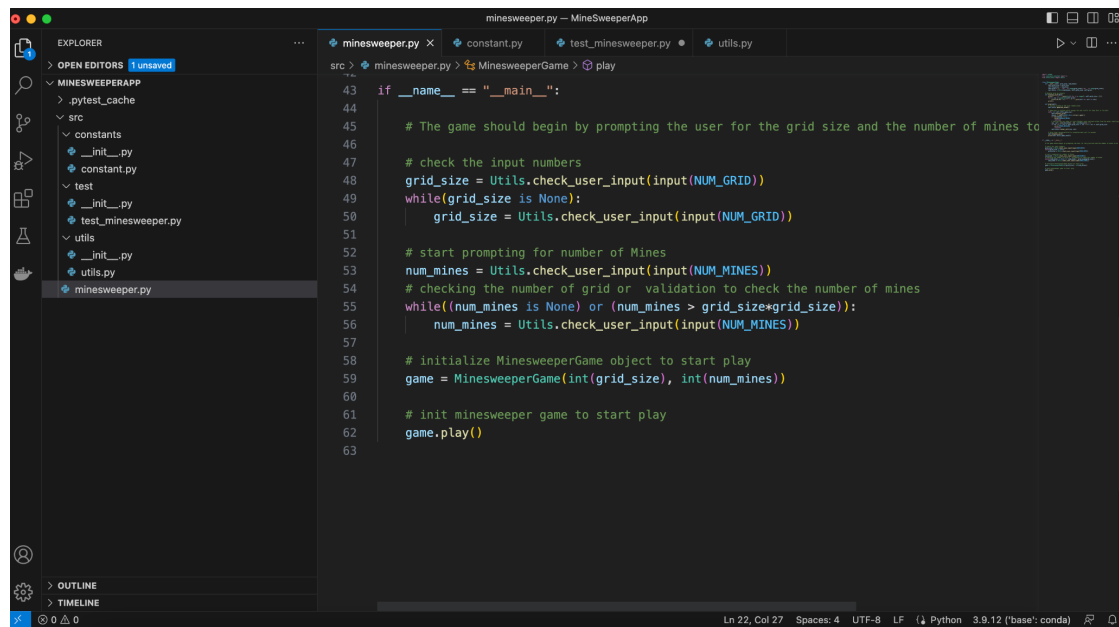


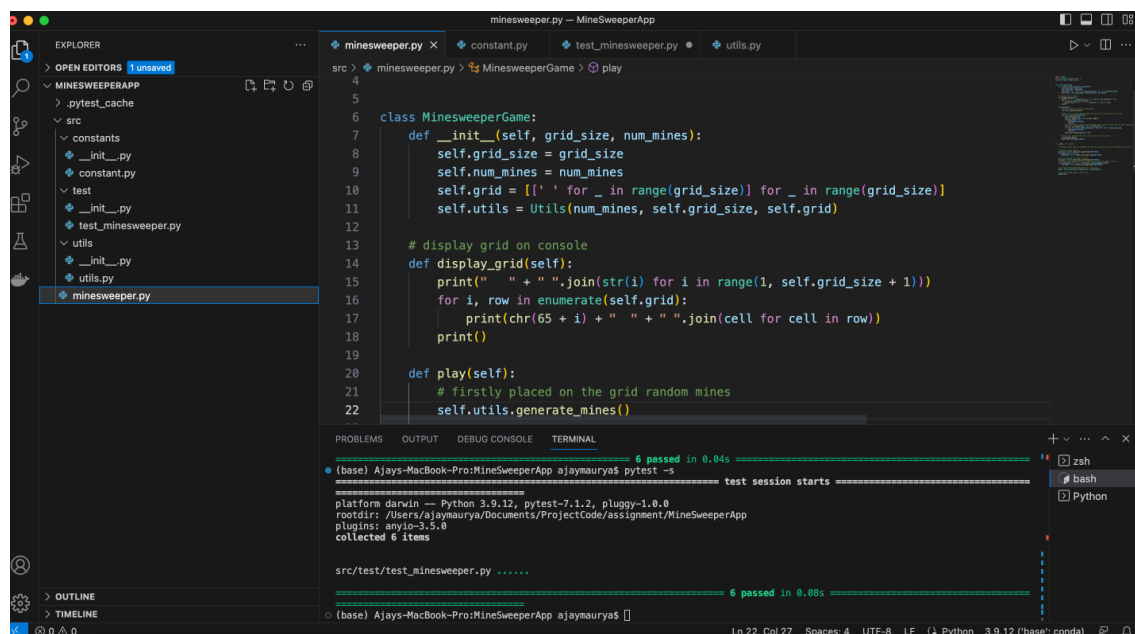
MineSweeper Game Application

Brief Explanations: Minesweeper is a game on a Square grid matrix application that I have developed from scratch in **Python Programming Language** and demonstrated a good understanding of clean code in a modular approach, fully object-oriented programming based on class and object, and the SOLID principle of the software development lifecycle. Also, Included **PyTest BDD** to solve this MinSweeper Game application.

Also, I have created a package folder to maintain the handlers and constants in the specific folder structure. Please refer to the screenshot below.



```
src > minesweeper.py > MinesweeperGame > play
43 if __name__ == "__main__":
44
45     # The game should begin by prompting the user for the grid size and the number of mines to
46
47     # check the input numbers
48     grid_size = Utils.check_user_input(input(NUM_GRID))
49     while(grid_size is None):
50         grid_size = Utils.check_user_input(input(NUM_GRID))
51
52     # start prompting for number of Mines
53     num_mines = Utils.check_user_input(input(NUM_MINES))
54     # checking the number of grid or validation to check the number of mines
55     while((num_mines is None) or (num_mines > grid_size*grid_size)):
56         num_mines = Utils.check_user_input(input(NUM_MINES))
57
58     # initialize MinesweeperGame object to start play
59     game = MinesweeperGame(int(grid_size), int(num_mines))
60
61     # init minesweeper game to start play
62     game.play()
63
```



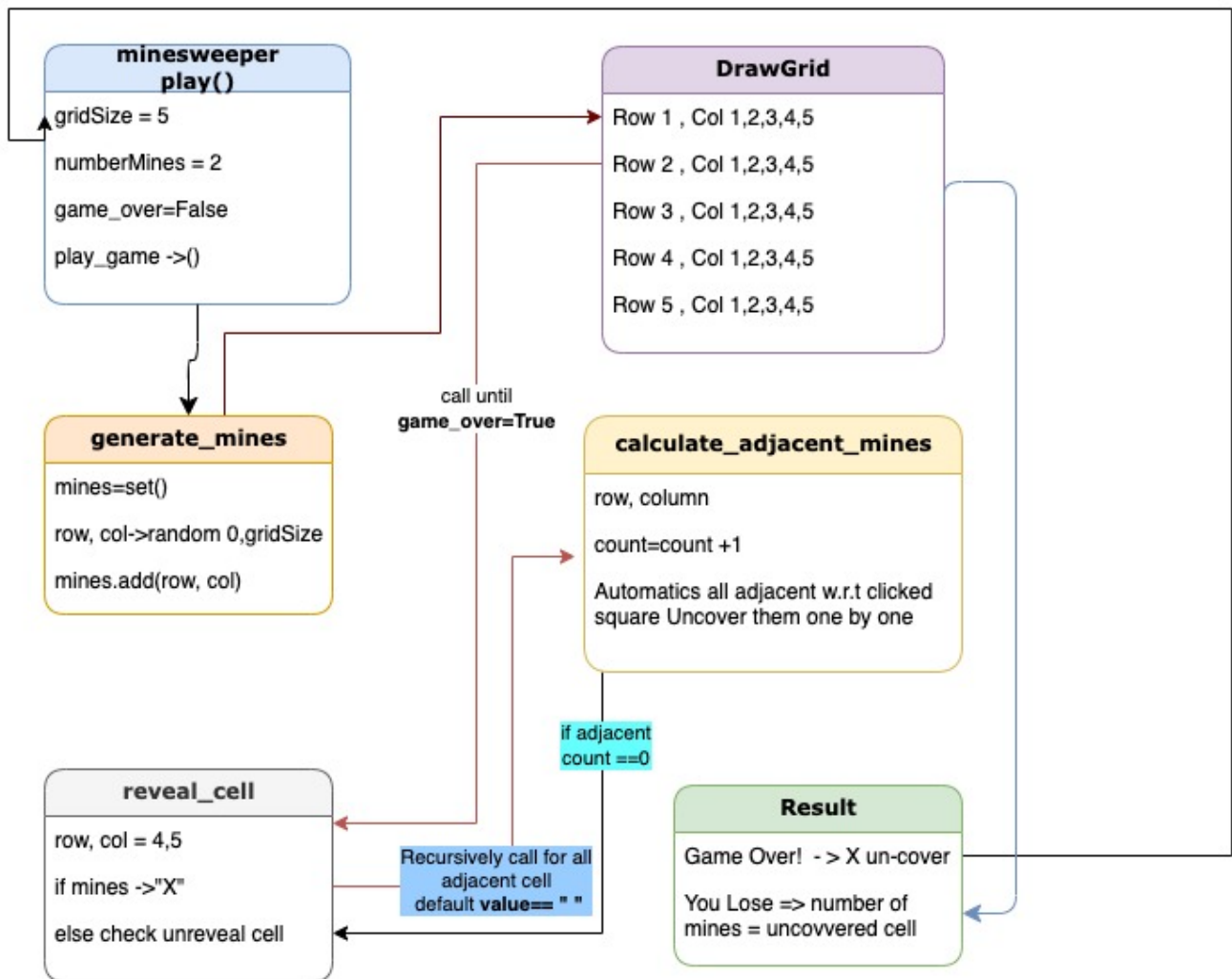
```
src > minesweeper.py > MinesweeperGame > play
4
5
6 class MinesweeperGame:
7     def __init__(self, grid_size, num_mines):
8         self.grid_size = grid_size
9         self.num_mines = num_mines
10        self.grid = [[' ' for _ in range(grid_size)] for _ in range(grid_size)]
11        self.utils = Utils(num_mines, self.grid_size, self.grid)
12
13        # display grid on console
14        def display_grid(self):
15            print(" " + " ".join(str(i) for i in range(1, self.grid_size + 1)))
16            for i, row in enumerate(self.grid):
17                print(chr(65 + i) + " " + " ".join(cell for cell in row))
18            print()
19
20        def play(self):
21            # firstly placed on the grid random mines
22            self.utils.generate_mines()
```

```
(base) Ajays-MacBook-Pro:MineSweeperApp ajaymaurya$ pytest -s
platform darwin -- Python 3.9.12, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/ajaymaurya/Documents/ProjectCode/assignment/MineSweeperApp
plugins: anyio-3.5.0
collected 6 items

src/test/test_minesweeper.py .....
===== 6 passed in 0.04s =====
test session starts
platform darwin -- Python 3.9.12, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/ajaymaurya/Documents/ProjectCode/assignment/MineSweeperApp
plugins: anyio-3.5.0
collected 6 items

src/test/test_minesweeper.py .....
===== 6 passed in 0.08s =====
```

DESIGN and ASSUMPTIONS: I have created some steps to cover the implementations as follows.



Note* Refer below to set up the project on a local machine and run the application

Step 1: Create **minesweeper.py** class to prompt the number of grid sizes and number of mines in terminal input.

```

43 if __name__ == "__main__":
44
45     # The game should begin by prompting the user for the grid size and the number of mines to be randomly p
46
47     # check the input numbers
48     grid_size = Utils.check_user_input(input(NUM_GRID))
49     while(grid_size is None):
50         grid_size = Utils.check_user_input(input(NUM_GRID))
51
52     # start prompting for number of Mines
53     num_mines = Utils.check_user_input(input(NUM_MINES))
54     # checking the number of grid or validation to check the number of mines
55     while((num_mines is None) or (num_mines > grid_size*grid_size)):
56         num_mines = Utils.check_user_input(input(NUM_MINES))
57
58     # initialize MinesweeperGame object to start play
59     game = MinesweeperGame(int(grid_size), int(num_mines))
60
61     # init minesweeper game to start play
62     game.play()
63

```

```

/MineSweeperApp/src/minesweeper.py
Enter Grid Size: 5
Enter No. of Mines: 2
1 2 3 4 5
A
B
C
D
E
Enter Cell Input (Like-> A1):

```

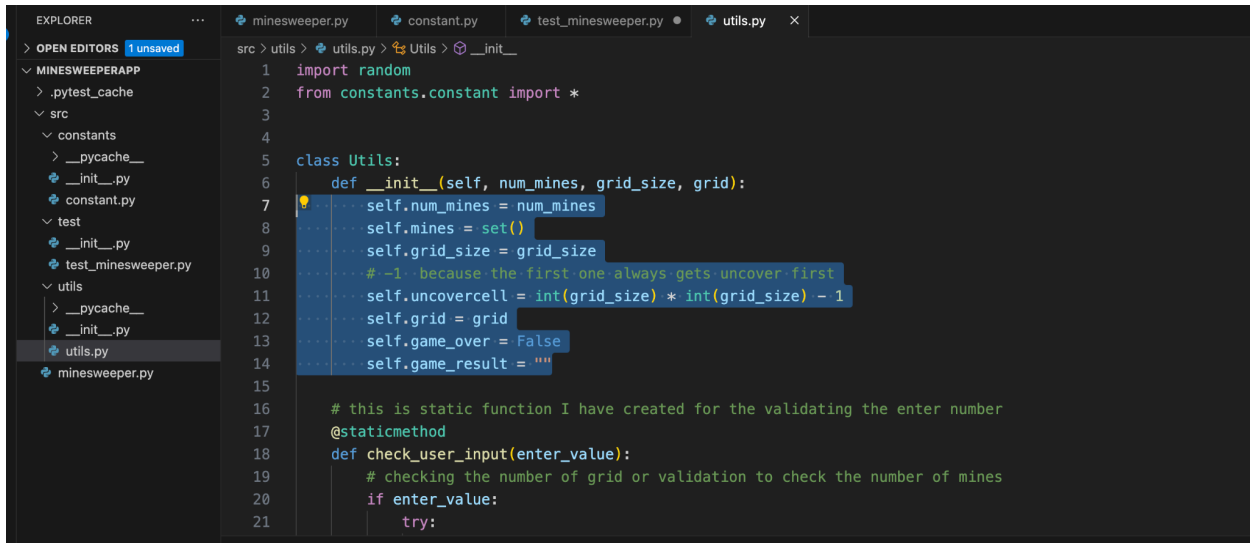
Step 2: Create **minesweeper.py** class that holds the value of **grid_size**, the number of **mines**. In this step itself, based on the grid size I created a matrix array with the entered size variable name **"grid"**. In the **Utils** object implemented main business logic about the **reveal_cell** and **game over** and **game_result**. **Based on the above entries it 5X5 grid and 2 Mines**

```

2 from constants.constant import *
3 from utils.utils import Utils
4
5
6 class MinesweeperGame:
7     def __init__(self, grid_size, num_mines):
8         self.grid_size = grid_size
9         self.num_mines = num_mines
10        self.grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
11        self.utils = Utils(num_mines, self.grid_size, self.grid)
12
13    # display grid on console
14    def display_grid(self):
15        print("\n\n")
16        for i in range(self.grid_size):
17            for j in range(self.grid_size):
18                print(self.grid[i][j], end=" ")
19            print("\n")

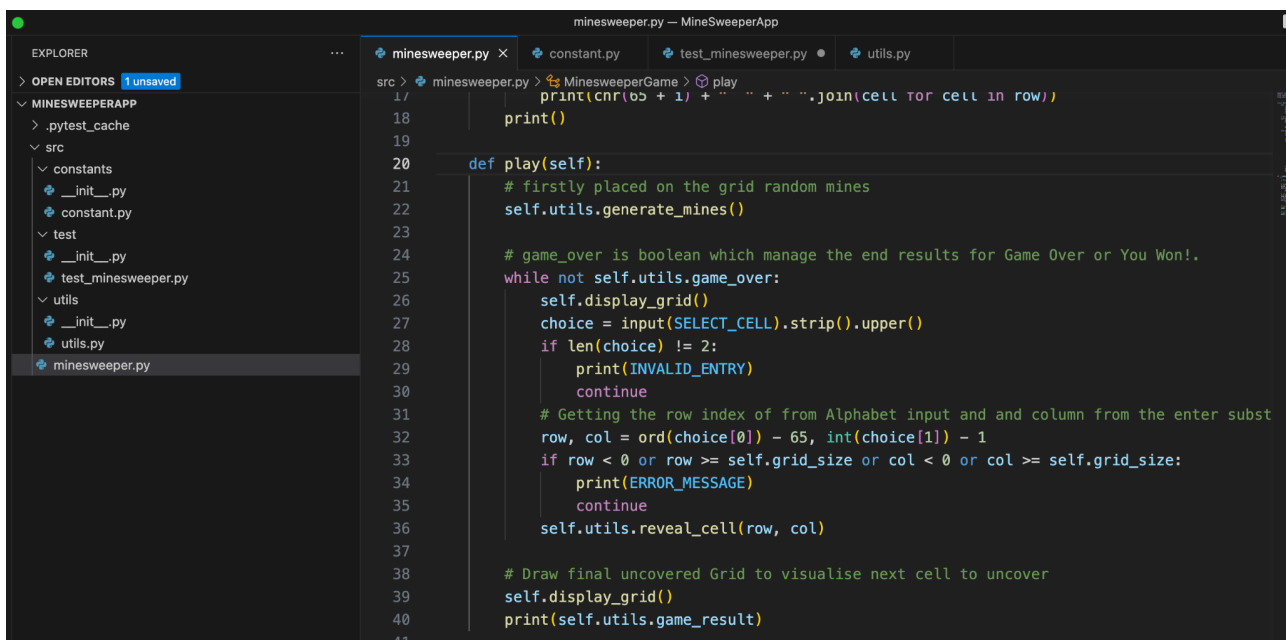
```

Utils: Initializing the **Utils** class by defining the required variable and game status.



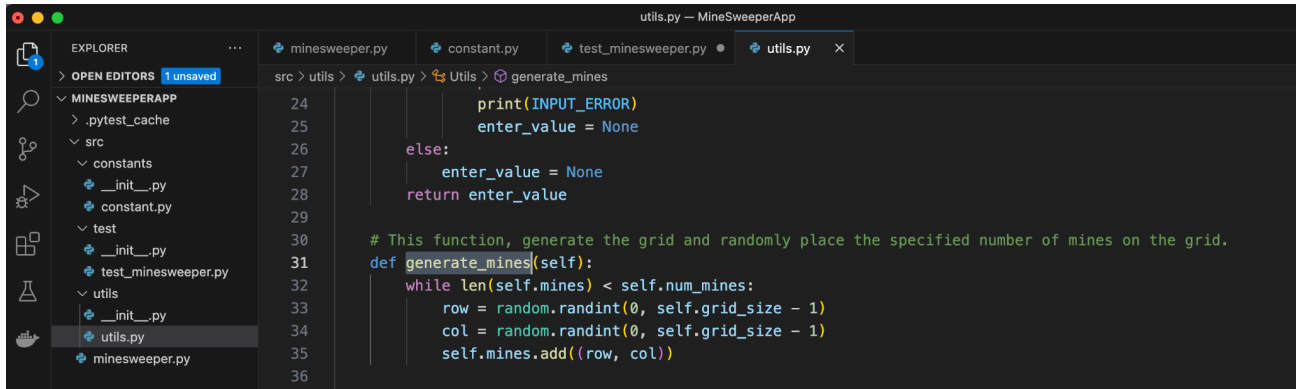
```
src > utils > utils.py > Utils > __init__
1  import random
2  from constants.constant import *
3
4
5  class Utils:
6      def __init__(self, num_mines, grid_size, grid):
7          self.num_mines = num_mines
8          self.mines = set()
9          self.grid_size = grid_size
10         # -1 because the first one always gets uncover first
11         self.uncovercell = int(grid_size) * int(grid_size) - 1
12         self.grid = grid
13         self.game_over = False
14         self.game_result = ""
15
16         # this is static function I have created for the validating the enter number
17         @staticmethod
18         def check_user_input(enter_value):
19             # checking the number of grid or validation to check the number of mines
20             if enter_value:
21                 try:
```

Step 3: Then trigger the play game function which is created in this class **minesweeper.py**. It will



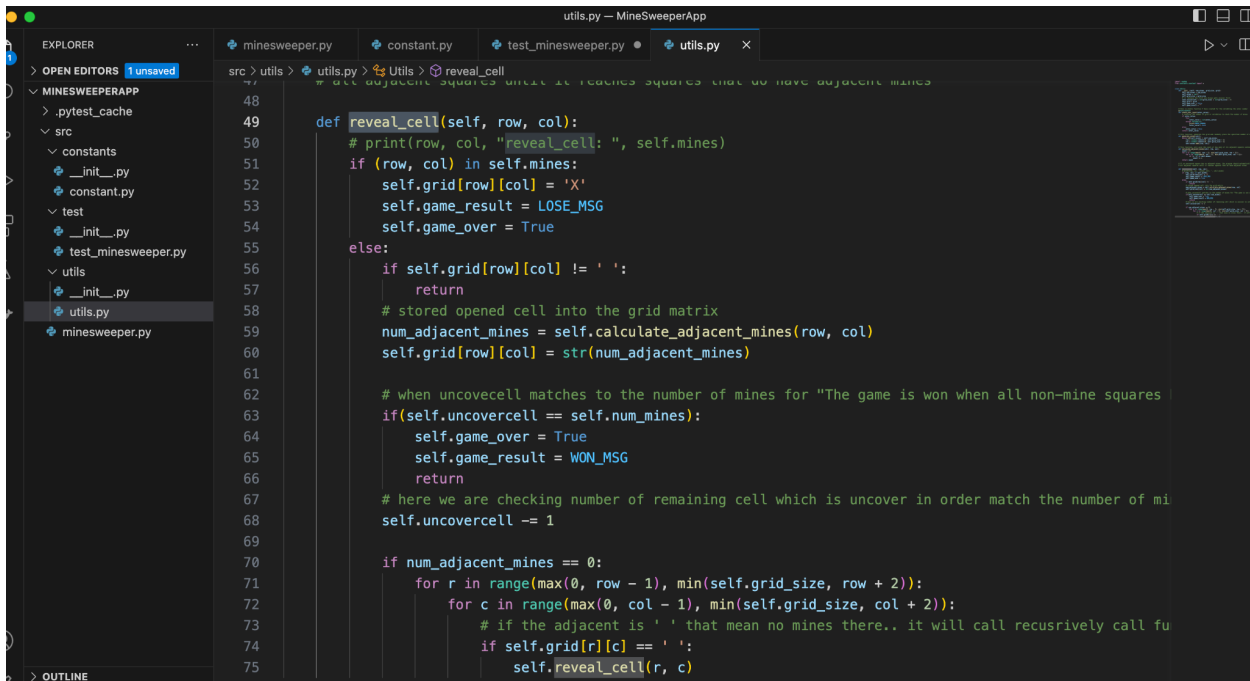
```
minesweeper.py — MinesweeperApp
src > minesweeper.py > MinesweeperGame > play
17  print(cnr(b5 + 1) + " " + " ".join(cell for cell in row))
18  print()
19
20  def play(self):
21      # firstly placed on the grid random mines
22      self.utils.generate_mines()
23
24      # game_over is boolean which manage the end results for Game Over or You Won!.
25      while not self.utils.game_over:
26          self.display_grid()
27          choice = input(SELECT_CELL).strip().upper()
28          if len(choice) != 2:
29              print(INVALID_ENTRY)
30              continue
31          # Getting the row index of from Alphabet input and and column from the enter subst
32          row, col = ord(choice[0]) - 65, int(choice[1]) - 1
33          if row < 0 or row >= self.grid_size or col < 0 or col >= self.grid_size:
34              print(ERROR_MESSAGE)
35              continue
36          self.utils.reveal_cell(row, col)
37
38          # Draw final uncovered Grid to visualise next cell to uncover
39          self.display_grid()
40          print(self.utils.game_result)
41
```

Step 4: In this step, based on the number input for mines, we will generate random places where mine in **utils.py** file.



```
src > utils > utils.py > Utils > generate_mines
24         print(INPUT_ERROR)
25         enter_value = None
26     else:
27         enter_value = None
28         return enter_value
29
30     # This function, generate the grid and randomly place the specified number of mines on the grid.
31     def generate_mines(self):
32         while len(self.mines) < self.num_mines:
33             row = random.randint(0, self.grid_size - 1)
34             col = random.randint(0, self.grid_size - 1)
35             self.mines.add((row, col))
36
```

Step 5: Select a specific square cell and call utils function “**reveal_cell**” if the cell “**X**” i.e. Mines, then you lose, else it will check the adjacent count to uncover all possible cells automatically.



```
src > utils > utils.py > Utils > reveal_cell
# get adjacent squares until it reaches squares that do have adjacent mines
48
49     def reveal_cell(self, row, col):
50         # print(row, col, "reveal_cell: ", self.mines)
51         if (row, col) in self.mines:
52             self.grid[row][col] = 'X'
53             self.game_result = LOSE_MSG
54             self.game_over = True
55         else:
56             if self.grid[row][col] != ' ':
57                 return
58             # stored opened cell into the grid matrix
59             num_adjacent_mines = self.calculate_adjacent_mines(row, col)
60             self.grid[row][col] = str(num_adjacent_mines)
61
62             # when uncovercell matches to the number of mines for "The game is won when all non-mine squares
63             if (self.uncovercell == self.num_mines):
64                 self.game_over = True
65                 self.game_result = WON_MSG
66                 return
67             # here we are checking number of remaining cell which is uncover in order match the number of mi
68             self.uncovercell -= 1
69
70             if num_adjacent_mines == 0:
71                 for r in range(max(0, row - 1), min(self.grid_size, row + 2)):
72                     for c in range(max(0, col - 1), min(self.grid_size, col + 2)):
73                         # if the adjacent is ' ' that mean no mines there.. it will call recursively call fu
74                         if self.grid[r][c] == ' ':
75                             self.reveal_cell(r, c)
76
```

Approach:

1. Check the min exists (X) in the place of the selected cell square. If yes then Game Over!
2. Else, it will be checked whether the cell is uncovered already or not. If not then I am calculating the number of mines in that row, col adjacent. Automatically, it will look for all other adjacent recursively by calling the function “**reveal_cell()**” if the cell squared **default value** is “ ”
3. If all adjacent uncovered and remaining adjacent are equal to the number of covered cells then the user will be a winner.

The environment required to run the application: In Windows/MacOS

Install Python3 on the local machine. Make sure in the local machine the python is installed.

Download and Install Python:

<https://www.python.org/downloads/>

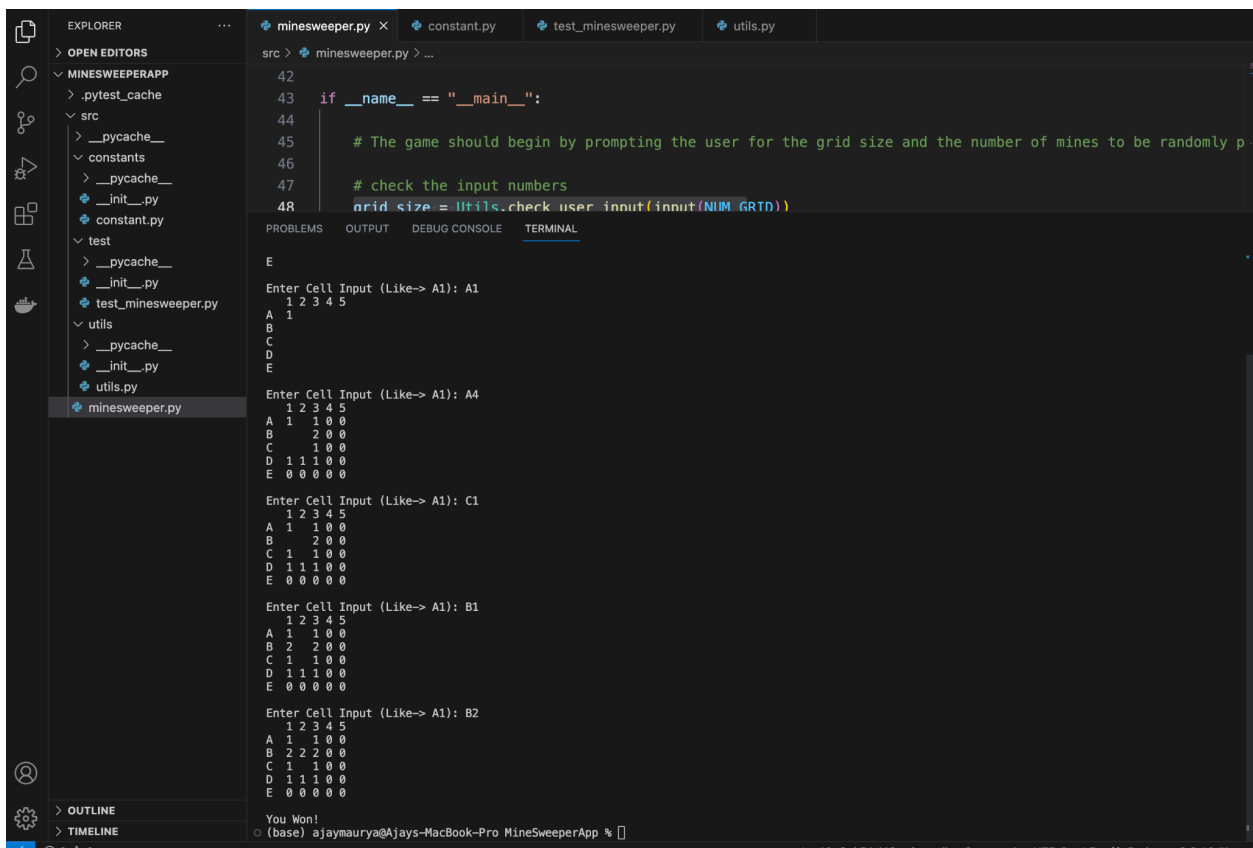
Make sure Python is installed using the below command.

> python -V

Python 3.8.4

Open Visual Studio Code or IDE which you prefer: Run the Application using the below command to start playing the minesweeper Game:

1. python minesweeper.py



The screenshot shows the Visual Studio Code interface with the 'minesweeper.py' file open in the editor. The file contains the following code:

```
42
43 if __name__ == "__main__":
44     # The game should begin by prompting the user for the grid size and the number of mines to be randomly p
45     # check the input numbers
46     grid_size = Utils.check_user_input(input(NUM_GRID))
```

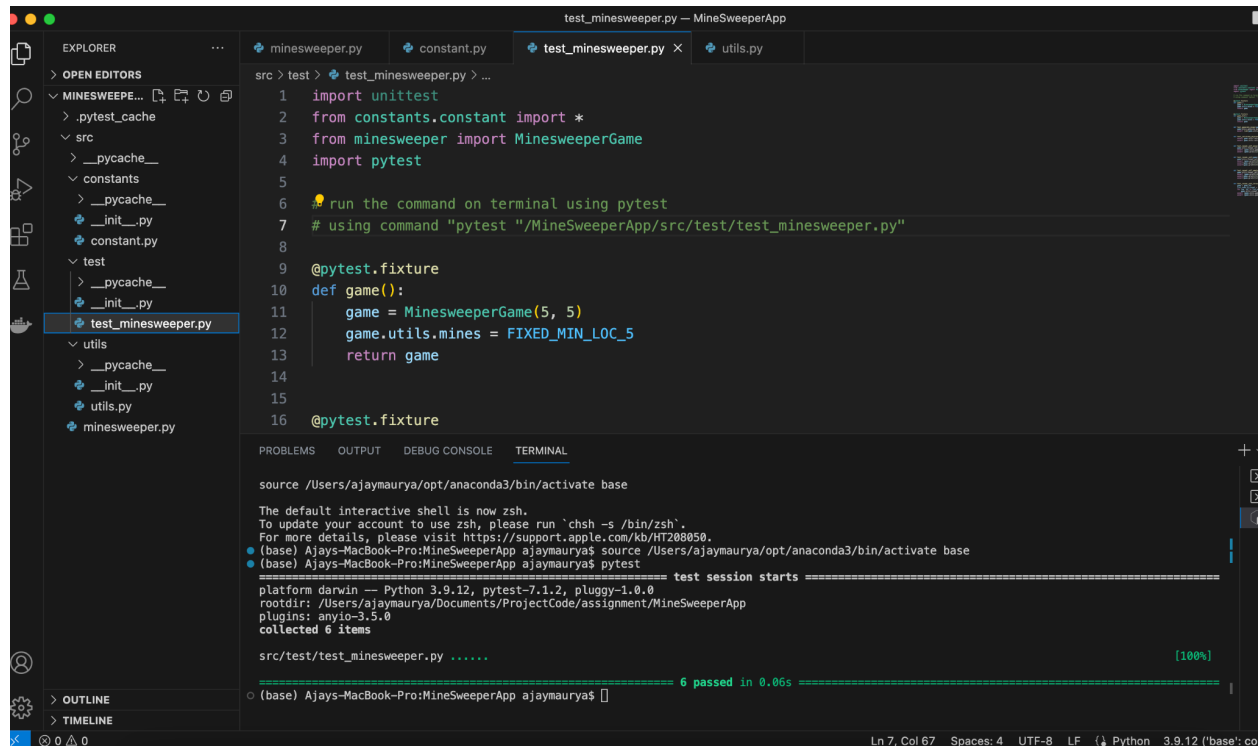
The terminal window shows the execution of the program. It prompts the user to enter a cell input (like A1) and displays the resulting grid state. The user enters 'A1', 'A4', 'C1', 'B1', and 'B2' in sequence. The grid state is updated after each input. The final state of the grid is:

```
1 2 3 4 5
A 1 1 0 0
B 2 2 0 0
C 1 1 0 0
D 1 1 1 0 0
E 0 0 0 0 0
```

The terminal also shows the message 'You Won!' and the command prompt '(base) ajaymaurya@Ajays-MacBook-Pro: MinesweeperApp %'.

TDD pyTest: Run the below command on the terminal it will automatically execute the created **test_minesweeper.py** class. Please refer below screenshot

> **pytest**



The screenshot shows a code editor with the file **test_minesweeper.py** open. The file contains the following code:

```
1 import unittest
2 from constants.constant import *
3 from minesweeper import MinesweeperGame
4 import pytest
5
6 # run the command on terminal using pytest
7 # using command "pytest "/MineSweeperApp/src/test/test_minesweeper.py"
8
9 @pytest.fixture
10 def game():
11     game = MinesweeperGame(5, 5)
12     game.utils.mines = FIXED_MIN_LOC_5
13     return game
14
15
16 @pytest.fixture
```

The terminal output shows the command **source /Users/ajaymaurya/opt/anaconda3/bin/activate base** and the command **pytest** being executed. The output indicates that the test session starts, and 6 items are collected. The test session passes, with 6 passed in 0.06s.

Note* How to Install pytest on local machine?

1. Type "cmd" in the search bar and hit Enter to open the command line.
2. Type "**pip install pytest**" (without quotes) in the command line and hit Enter again. This installs pytest for your default Python installation.
3. The previous command may not work if you have both **Python** versions **2** and **3** on your computer. In this case, try "**pip3 install pytest**" or "**python -m pip install pytest**".
4. Wait for the installation to terminate successfully. It is now installed on your Windows machine