FACULTY OF COMPUTING
& INFORMATION SCIENCES

جامعة مصر للمعلوماتية
EGYPT UNIVERSITY
OF INFORMATICS

Introduction to Computing & Programming

C-CS111

Fall 2023

Bingo Game

Team Members

Student Name	Student ID
Marwan Mostafa	23-101165
Abdelrahman Ahmed	23-101010
Abdelrahman Amr	23-101225
Karim Wesam	23-101254

Contents

Abstract

This comprehensive report establishes the precise planning and successful development of a sophisticated computer software system designed for a complete Bingo gaming experience. The complete program, which is executed precisely, is written in Python, a versatile programming language that makes use of abstract data types and modular design. To ensure a fluid and captivating user experience, great care was taken to enhance all aspects of game logic during the development phase, especially during computer turns and moves.

The Bingo game unfolds within a well-organized and efficient structure, allowing players to enjoy a captivating blend of chance and strategy. The implementation of modular design principles enables easy extensibility and maintainability, contributing to the durability and scalability of the software. The codebase's efficiency and clarity are improved by the underlying abstract data types, which make it easier to manipulate complicated game states.

This project demonstrates an organized approach to software development, showcasing the complex interaction between design concepts and algorithms. The report explores the theoretical foundations of the code and includes a user manual that gives gamers a thorough rundown of how to use all the system's functions. The user manual ensures a simple and delightful user experience by explaining important features, providing helpful hints and techniques, and outlining scenarios with screenshots.

Furthermore, the paper summarizes the key choices made in terms of design, coding conventions, and difficulties that arose during the development process. This Bingo game project is an excellent example of how programming concepts can be seamlessly integrated from conception to execution, which helps the project team get a better understanding of software development techniques.

In conclusion, this report not only serves as a testament to the successful creation of a Bingo game but also stands as a valuable resource for enthusiasts, programmers, and learners seeking insights into effective software design and development practices

1. Introduction

- The Bingo Game Project is a Python implementation. The game involves two players, user and the computer, each with their own game boards. The objective is to cross out numbers on the board and achieve specific patterns to win the game. It has multiple functions from initializing the boards and handling user inputs to checking for wins and eventually outsmarting the user.

2. Motivation

- The motivation behind selecting this problem was to create an engaging game that challenges our abilities in integrating user engagement and strategic decision-making. The project serves as a showcase for implementing diverse functionalities, encircling the management of user inputs, turn transitions, and computer-driven moves.

3. System Design

➢ System modules and Functions

- **get_random_message(key):**
  - **Input: key** (string)
  - **Output:** Random message (string)
  - **Functionality:** Returns a random message from the predefined dictionary based on the given key.
- **set_difficulty():**
  - **Input:** None (user input)
  - **Output:** Difficulty level (float)
  - **Functionality:** Allows the user to choose the difficulty level and returns the corresponding difficulty factor.
- **set_dimension():**
  - **Input:** None (user input)
  - **Output:** Tuple of user and computer boards (2d list)
  - **Functionality:** Asks the user for board dimensions and returns two randomized boards.
  - Nested Function:
    - **create_board():**
      - **Input:** None
      - **Output:** Randomized game board (2d list)
      - **Functionality:** Generates a randomized game board based on the specified dimensions.
- **play_game(name, difficulty, crossed_dict, user_board, comp_board):**
  - **Input:**
    - **name** (string): Player's name
    - **difficulty** (float): Difficulty level
    - **crossed_dict** (dictionary): Dictionary to track crossed positions in computer board only (used in the computer's smart moves)

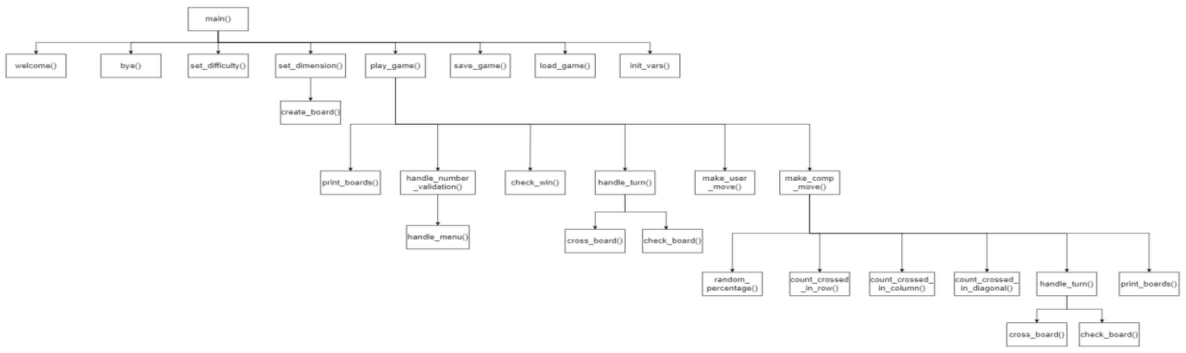
        - **user_board** (2d list): User's board
        - **comp_board** (2d list): Computer's board
    - **Output:** Boolean indicating whether the game is finished
    - **Functionality:** Manages the gameplay, including user and computer moves, checking for wins, and displaying the current game state.
    - Nested Functions:
        - **print_boards(user_board, comp_board):**
            - **Input:** User's and computer's boards
            - **Output:** None
            - **Functionality:** Prints the current state of user and computer boards.
        - **handle_number_validation():**
            - **Input:** None (user input)
            - **Output:** Validated user input (integer)
            - **Functionality:** Handles user input validation for selecting a number or accessing the menu.
        - **check_win():**
            - **Input:** None
            - **Output:** None
            - **Functionality:** Checks for a win condition and updates game variables accordingly.
        - **handle_turn(chosen_number, comp_turn):**
            - **Input:**
                - **chosen_number** (integer): The selected number
                - **comp_turn** (boolean): Indicates whether it's the computer's turn
            - **Output:** None
            - **Functionality:** Handles the turn logic, crosses the chosen number on the board, and checks for a win.
        - **make_user_move():**
            - **Input:** None (user input)
            - **Output:** Boolean indicating whether the user chose to quit
            - **Functionality:** Handles the user's move during the game.
        - **make_comp_move():**
            - **Input:** None
            - **Output:** None
            - **Functionality:** Handles the computer's move during the game, with varying levels of difficulty.
            - Nested Functions:
                - **random_percentage(percent):**
                    - **Input: percent** (float): Percentage value

            - **Output:** Boolean indicating whether a random value is less than or equal to the given percentage
            - **Functionality:** Generates a random float and compares it with the provided percentage.
        - **count_crossed_in_row(row_index):**
            - **Input: row_index** (integer): Index of the row to count crossed positions
            - **Output:** Number of crossed positions in the specified row
            - **Functionality:** Counts the number of crossed positions in the given row.
        - **count_crossed_in_column(col_index):**
            - **Input: col_index** (integer): Index of the column to count crossed positions
            - **Output:** Number of crossed positions in the specified column
            - **Functionality:** Counts the number of crossed positions in the given column.
        - **count_crossed_in_diagonal(is_main):**
            - **Input: is_main** (boolean): Indicates whether to count crossed positions in the main diagonal
            - **Output:** Number of crossed positions in the specified diagonal
            - **Functionality:** Counts the number of crossed positions in the main or secondary diagonal.
- **save_game(file_name, difficulty=None, crossed_dict={}, user_board=[], comp_board=[]):**
    - **Input:**
        - **file_name** (string): Name of the save file
        - **difficulty** (float, optional): Difficulty level
        - **crossed_dict** (dictionary, optional): Dictionary to track crossed positions
        - **user_board** (2d list, optional): User's board
        - **comp_board** (2d list, optional): Computer's board
    - **Output:** None
    - **Functionality:** Saves the game state and related information to a file.
    - Nested Function:
        - **write_board(board):**
            - **Input:** Game board (2d list)
            - **Output:** None

        - **Functionality:** Writes the content of a board to the save file.

- **load_game():**
    - **Input:** None (user input)
    - **Output:** Tuple of game information (name, difficulty, crossed_dict, user_board, comp_board)
    - **Functionality:** Loads a saved game from a file.
- **init_vars():**
    - **Input:** None
    - **Output:** Tuple of initialized variables (user_won, user_score, comp_score, user_board, comp_board)
    - **Functionality:** Initializes game-related global variables.
- **main(name=None):**
    - **Input: name** (string, optional): Player's name
    - **Output:** None
    - **Functionality:** The main function orchestrating the overall game flow, including starting a new game, loading a saved game, and handling player moves.

- Hierarchy Chart



4. Description

- The Bingo Game Project, developed in Python, goes beyond the conventional implementation by introducing a sophisticated game environment that strategically challenges users. Through diligent design and coding, the project aims to provide an engaging experience where the computer intelligently competes with the user. It encompasses a range of functions designed to handle diverse aspects of the game, such as managing the game state, processing user input, and executing strategic computer moves.

- One notable feature of the project is the implementation of a game flow that seamlessly transitions between user and computer turns, creating a dynamic and competitive atmosphere. The computer's moves are not merely random; instead, an intelligent decision-making process is employed to outsmart the human player. The algorithm selects optimal spots on the board, showcasing the project's emphasis on strategic depth and artificial intelligence.

- The implementation also reinforces the versatility of Python, showcasing its capabilities in handling file operations for saving and loading game states. The project's core logic is built around the mechanics of Bingo, making it a comprehensive demonstration of game development, logical reasoning, and interactive programming. In summary, the Bingo Game Project is not just a conventional game implementation but a showcase of advanced functionalities, where the computer's intelligence adds an extra layer of challenge, creating a compelling and strategic gaming experience. The project is a testament to the capabilities of Python in game development and serves as an excellent example of merging user interaction with sophisticated programming logic.

5. Team’s Workload Distribution

- **Modules**:

1. Game Logic: turns, computer smart moves, wins check, difficulties, crossing and checking boards, and input validation.

2. File Handling: Save and Load

3. Testing and Debugging

4. User Interface: Print boards, messages

5. Documentation: Report writing, comments

Name	**Responsibilities**
Marwan Mostafa	**Module 1, Module 3, Module 5**
Abdelrahman Ahmed	**Module 1 and Module 5**

Abdelrahman Amr	**Module 2**
Karim Wesam	**Module 4**

6. User Manual

To start the game, run the file named “main.py”. Follow the instructions to start, from entering your name and choosing the dimensions of the boards. Remember that your name will be the name of your saved file.

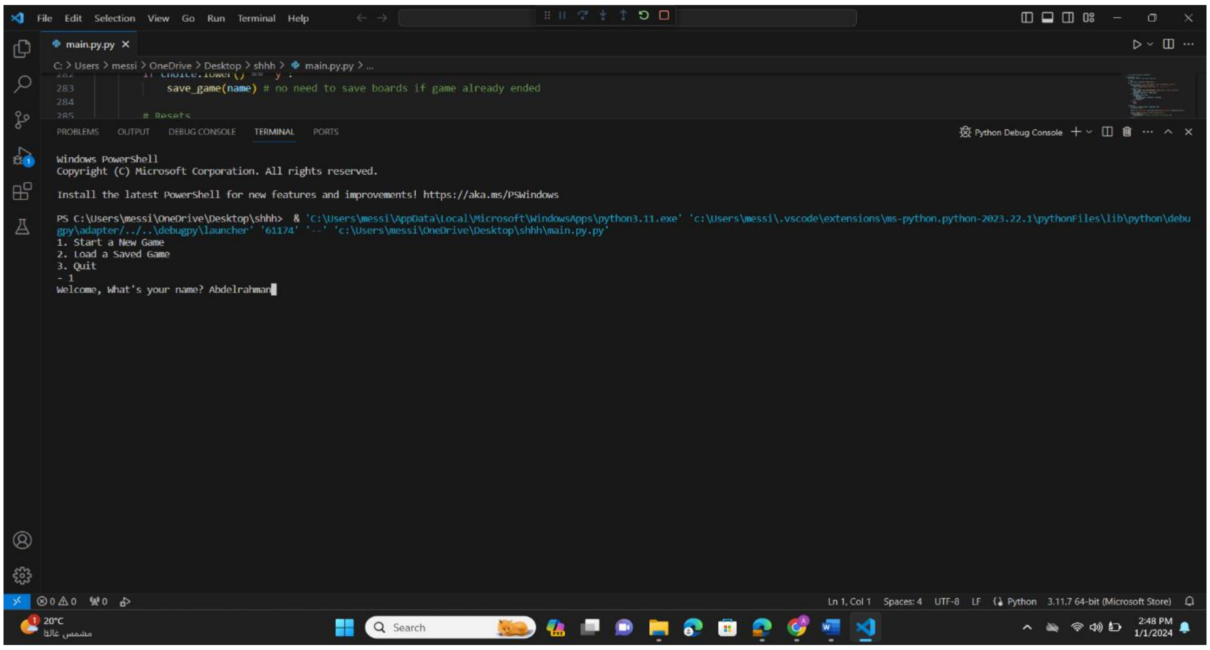
Moreover, check the readme for the video link on how to play.

Some shortcuts:

- Y for Yes
- N for No
- M for menu
- And sometimes there will be options to choose from (1, 2, 3, ..).

Sample Screens:

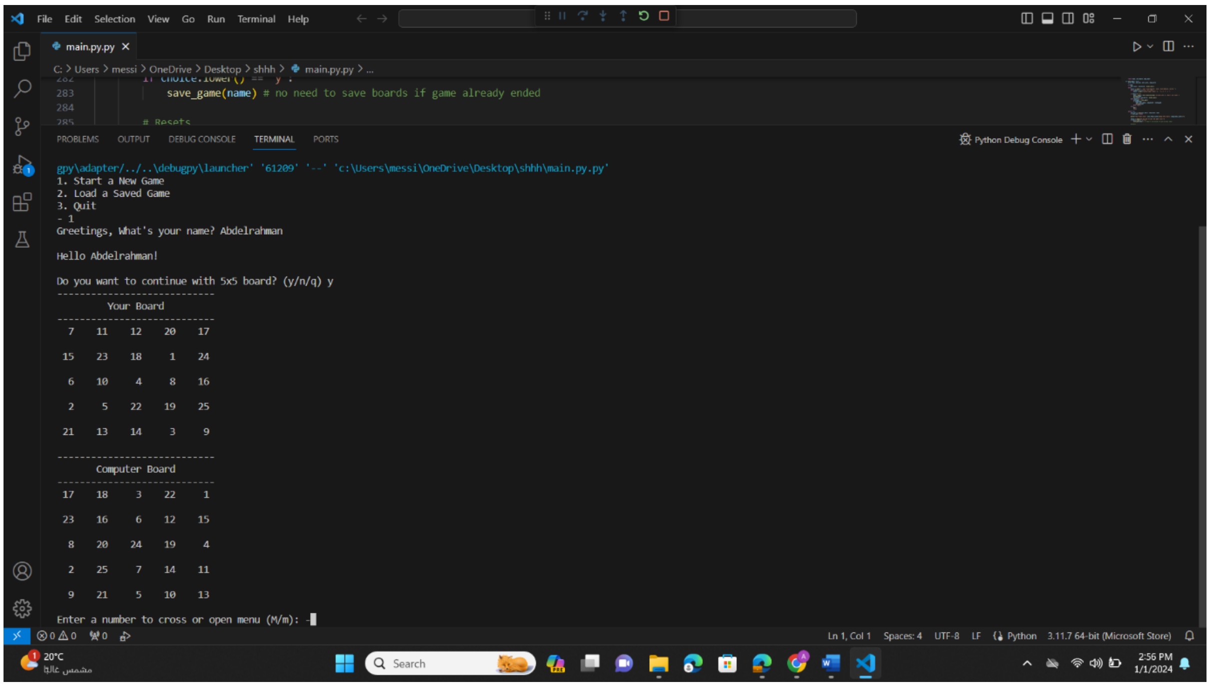
Here the user is choosing one of three options and of course the user will click a new game if it is his first time. After that the user must enter his name (the same name of his save file) to continue:



```
            save_game(name) # no need to save boards if game already ended

        # Resets
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\messi\OneDrive\Desktop\shhh>  & 'C:\Users\messi\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\messi\.vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '61174' '--' 'c:\Users\messi\OneDrive\Desktop\shhh\main.py.py'
1. Start a New Game
2. Load a Saved Game
3. Quit
- 1
Welcome, What's your name? Abdelrahman
```

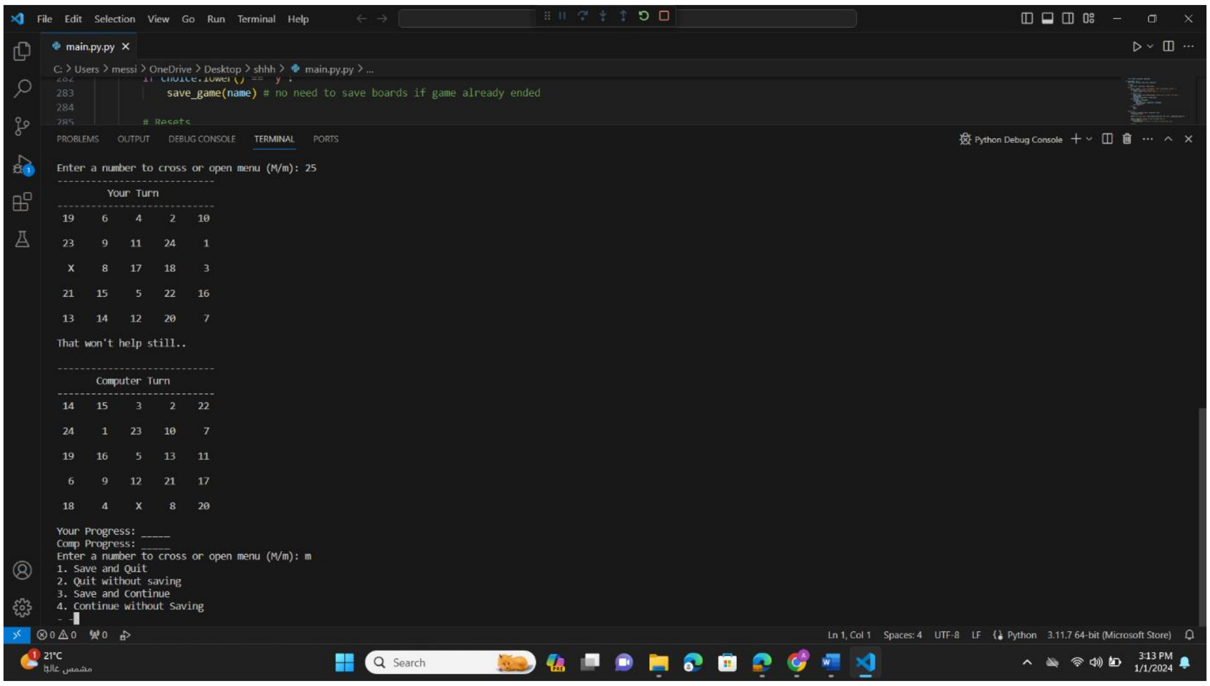
The program asks if the user wants to continue with the standard 5x5 board, if no (n), the user must enter a custom dimension:



```
            save_game(name) # no need to save boards if game already ended

        # Resets
```

```
gpy\adapter/../..\debugpy\launcher' '61209' '--' 'c:\Users\messi\OneDrive\Desktop\shhh\main.py.py'
1. Start a New Game
2. Load a Saved Game
3. Quit
- 1
Greetings, What's your name? Abdelrahman

Hello Abdelrahman!

Do you want to continue with 5x5 board? (y/n/q) y
-----------------------------
         Your Board
-----------------------------
  7   11   12   20   17

 15   23   18    1   24

  6   10    4    8   16

  2    5   22   19   25

 21   13   14    3    9

-----------------------------
       Computer Board
-----------------------------
 17   18    3   22    1

 23   16    6   12   15

  8   20   24   19    4

  2   25    7   14   11

  9   21    5   10   13

Enter a number to cross or open menu (M/m): -
```

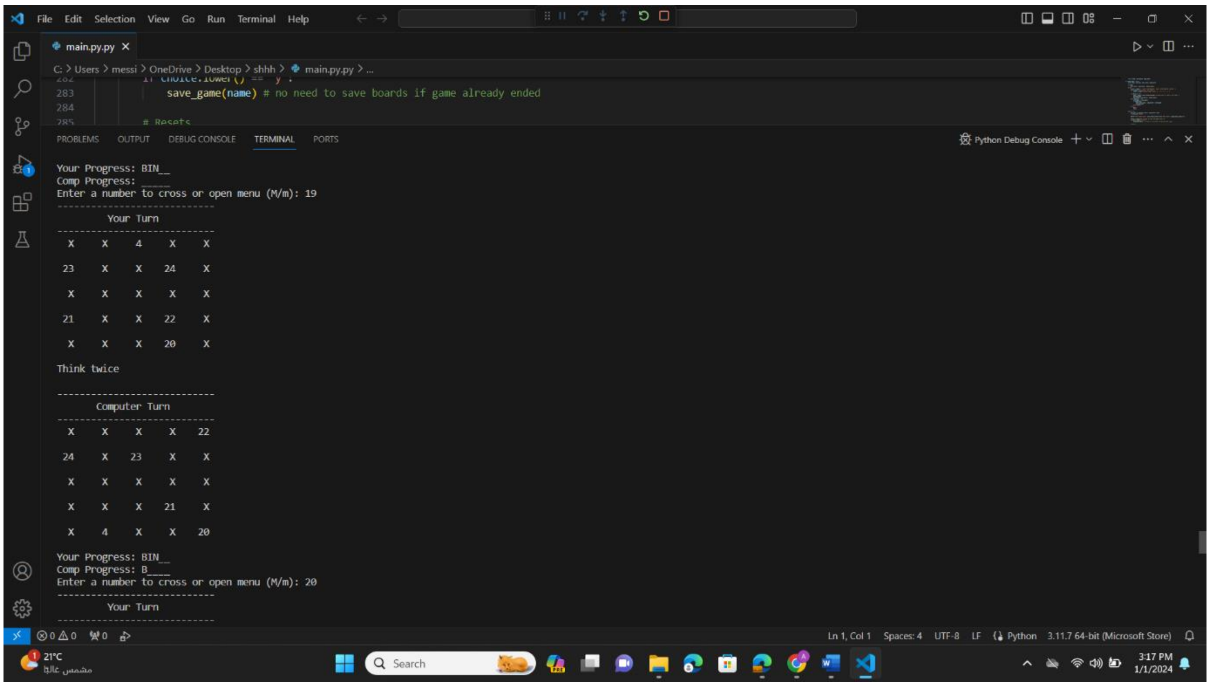
The user enters a number to be crossed out and has the option to save the game or quit:



```
        save_game(name) # no need to save boards if game already ended
```

```
Enter a number to cross or open menu (M/m): 25
-------------------------
        Your Turn
-------------------------
 19     6     4     2    10

 23     9    11    24     1

  X     8    17    18     3

 21    15     5    22    16

 13    14    12    20     7

That won't help still..

-------------------------
      Computer Turn
-------------------------
 14    15     3     2    22

 24     1    23    10     7

 19    16     5    13    11

  6     9    12    21    17

 18     4     X     8    20

Your Progress: _____
Comp Progress: _____
Enter a number to cross or open menu (M/m): m
1. Save and Quit
2. Quit without saving
3. Save and Continue
4. Continue without Saving
- 
```

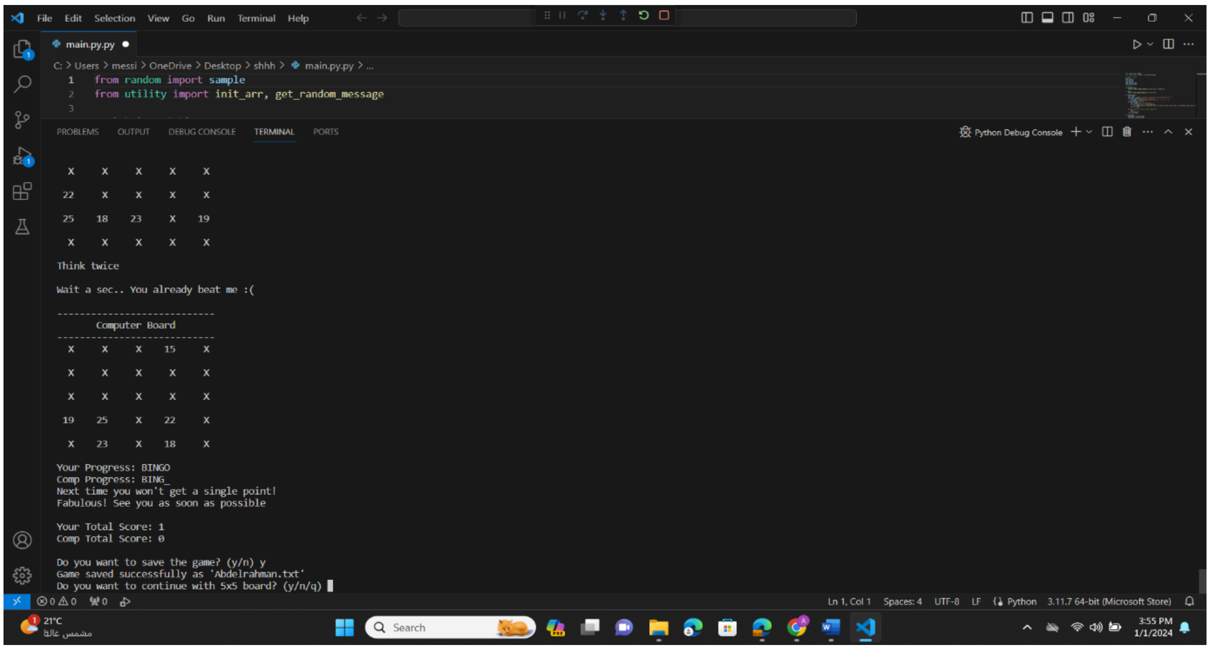
After a row, column, or a diagonal is crossed the user uncovers a letter from the word BINGO and so on until the user completes the word:



```
        save_game(name) # no need to save boards if game already ended
```

```
Your Progress: BIN__
Comp Progress: _____
Enter a number to cross or open menu (M/m): 19
-------------------------
        Your Turn
-------------------------
  X     X     4     X     X

 23     X     X    24     X

  X     X     X     X     X

 21     X     X    22     X

  X     X     X    20     X

Think twice

-------------------------
      Computer Turn
-------------------------
  X     X     X     X    22

 24     X    23     X     X

  X     X     X     X     X

  X     X     X    21     X

  X     4     X     X    20

Your Progress: BIN__
Comp Progress: B____
Enter a number to cross or open menu (M/m): 20
-------------------------
        Your Turn
-------------------------
```

User won by completing the word BINGO, then saved the game to a file with his name by pressing y (for yes):



```python
from random import sample
from utility import init_arr, get_random_message
```

```
  X    X    X    X    X

 22    X    X    X    X

 25   18   23    X   19

  X    X    X    X    X

Think twice

Wait a sec.. You already beat me :(

--------------------------
      Computer Board
--------------------------
  X    X    X   15    X

  X    X    X    X    X

  X    X    X    X    X

 19   25    X   22    X

  X   23    X   18    X

Your Progress: BINGO
Comp Progress: BING_
Next time you won't get a single point!
Fabulous! See you as soon as possible

Your Total Score: 1
Comp Total Score: 0

Do you want to save the game? (y/n) y
Game saved successfully as 'Abdelrahman.txt'
Do you want to continue with 5x5 board? (y/n/q)
```

File Edit Selection View Go Run Terminal Help
main.py.py
C: > Users > messi > OneDrive > Desktop > shhh > main.py.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.11.7 64-bit (Microsoft Store)
21°C
مشمس غالبًا
Search
3:55 PM
1/1/2024

7. Testing and Validation

1. **Valid Input Test:**
    - User chooses to start a new game.
    - User enters a valid name.
    - User and computer boards are initialized correctly.
2. **Invalid Dimension Input Test:**
    - User chooses a custom dimension and enters an invalid value.
    - The program prompts the user until a valid dimension is provided.
3. **Winning Scenario Test:**
    - Simulate a winning scenario for the user.
    - Ensures the game ends and the correct winner message is displayed.
4. **Saving and Loading Test:**
    - Save a game any time during the game and continues.
    - Ensures the loaded game has its correct states and variables.
5. **User Quit Test:**
    - User chooses to quit during their turn.
    - Ensures the program returns to the main menu and quits without saving the game.
6. **Computer's Best Move:**
    - Computer's turn started.
    - Computer checks each row, column, and diagonal.
    - Computer checks the best option in each row, column, and diagonal.
    - Computer retrieves the indices of the available spots and chooses based on priority

- Computer chooses to cross the spot that is the closest to gaining a point than the other best options.

7. **Difficulty Test:**
    - User chooses multiple difficulties.
    - In each computer turn the difficulty impacts how the computer responds based on random percentages.
    - Computer either chooses to play a smart move or to choose randomly based on the calculated percentages

8. Challenges and Conclusions

- **Challenges**:
    - The system was successful enough, and we fixed all kinds of problems and bugs that we encountered along the way.
    - This game made us learn a lot of skills like problem solving.
    - We faced a lot of challenges in:
        - The Printing System: Centering certain strings and displaying spaced-board items with consistent spacing between all messages and interactions between the user and the game.
        - Invalidating Inputs: was sometimes tricky, but we made it through. Causing some infinite loops of course.
        - Checking boards: The first logic I encountered was checking boards for a complete crossed row, column, or diagonal. Yet, not as hard as the upcoming one.
        - The Computer Smart Moves: Filtering out all available spots in each row, column, and diagonal, then choosing one spot in each depending on the priorities of each spot, which is essentially the number of crossed spots in this same specific row, column, or diagonal, ending up with three filtered spots in total, then making the decision on one based on their priorities.

- **Conclusion**:
    - The Bingo Game Project successfully implements a Bingo-like game with user and computer interaction. The project achieves its objectives and beyond, Providing an entertaining gaming experience. Suggestions for improvements and future work may include adding more features like completing the GUI and enhancing the game's complexity.
        - **Suggestions for Improvements/Future Work:**

- **Multiplayer Support:** Support for multiplayer mode, either local or online, which can extend the game's capabilities (requires the integration of threads and sockets).
- **AI Improvement:** Enhancing the computer's strategy in tackling the player which can make the game more challenging and enjoyable.
- **Start Round**: Make Computer and User turns randomly or depending on the difficulty, instead of always starting the round by the user.
- **Additional Bingo Patterns:** Introducing different Bingo patterns can add variety to the game.
- **Finish GUI:** We did indeed intend to create the graphical user interface for the game. But sadly, we ran out of time. It was harder implementing the GUI in week 4, instead of prioritizing it first, and then building the game on top of it. That was later recognized as a big mistake/roadblock, that in short time we didn't get to solve it.

References

Listing the latest ones.

- "Built-in Functions." *Python Documentation*, docs.python.org/3/library/functions.html#max.
- This Course's Textbook, Chapter 8 (self-study) named "More about strings".
- *Python Tutorial*. www.w3schools.com/python/default.asp.
- *How to Check if a File or Directory Exists in Python - Python Engineer*. www.python-engineer.com/posts/check-if-file-exists.
- GeeksforGeeks. "Python List Slicing." *GeeksforGeeks*, 4 July 2023, www.geeksforgeeks.org/python-list-slicing.
- "---." *Python Documentation*, docs.python.org/3/library/functions.html#sum.
- *Python Random Choice() Method*. www.w3schools.com/python/ref_random_choice.asp.
- GeeksforGeeks. "How to Break Out of Multiple Loops in Python." *GeeksforGeeks*, 24 Feb. 2023, www.geeksforgeeks.org/how-to-break-out-of-multiple-loops-in-python.
- GeeksforGeeks. "Difference Between and Is Operator in Python." *GeeksforGeeks*, 10 Jan. 2023, www.geeksforgeeks.org/difference-between-and-is-operator-in-python.
- ---. "Python random.sample Function." *GeeksforGeeks*, 29 Aug. 2018, www.geeksforgeeks.org/python-random-sample-function.
- "Tkinter Tutorials - AskPython." *AskPython*, www.askpython.com/python-modules/tkinter.
- Code First with Hala. "Tkinter Text Widget - Tkinter Tutorial for Beginners #6." *YouTube*, 27 June 2022, www.youtube.com/watch?v=P9whqKpw43I.