B.M.S COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

UNIX SHELL PROGRAMMING

Report on

CONNECT 4 GAME

Submitted in partial fulfillment of the requirements for AAT

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

KATHASAGARAM AISHWARYA	1BM22CS123
MAANAS SAJEEV	1BM22CS139
MAHIKA D	1BM22CS142

NAME OF THE GUIDE

Prof. SNEHA S BAGALKOT
Assistant Professor
Department of CSE

Prof. SNEHA P
Assistant Professor
Department of CSE

Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2023-2024

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, KATHASAGARAM AISHWARYA (1BM22CS123), MAANAS SAJEEV (1BM22CS139) and MAHIKA D (1BM22CS142) students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this AAT Project entitled "CONNECT 4 GAME" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester December - March 2024. We also declare that to the best of our knowledge and belief, the Project report is not from part of any other report by any other students.

Signature of the Candidates

Kathasagaram Aishwarya

(1BM22CS123)

Signature of the Candidates

Maanas Sajeev

(1BM22CS139)

Signature of the Candidates

Mahika D

(1BM22CS142)

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the AAT Project titled "**CONNECT 4 GAME"** has been carried out by KATAHSAGARAM AISHWARYA (1BM22CS123), MAANAS SAJEEV (1BM22CS139) and MAHIKA D (1BM22CS142) during the academic year 2023-2024.

Signature of the Faculty in Charge | Signature of the HoD

Prof. Sneha S Bagalkot Assistant Professor Department of CSE	Prof. Sneha P Assistant Professor Department of CSE	Dr. Jyothi S Nayak Professor and Head Department of CSE

Signature of the Examiners with date:

1. Internal Examiner Name and Signature: ……..

2. External Examiner Name and Signature: ……..

Abstract

This project presents a simple implementation of the classic Connect 4 game using the Bash scripting language. The game utilizes basic shell scripting constructs and the **zenity** utility for creating graphical dialog boxes to interact with the players. The implementation includes functions for initializing the game board, printing the board state, checking for a win condition, and handling player moves.

The Connect 4 game is played on a 6x7 grid, where two players take turns dropping marks X and O into columns. The objective is to connect four of one's own marks vertically, horizontally, or diagonally before the opponent does.

The script allows players to input their moves via graphical dialog boxes, displaying the current state of the game board after each move. Additionally, it detects winning combinations and announces the winner, providing an engaging and interactive gaming experience entirely within the Bash environment.

This project demonstrates the versatility of Bash scripting for implementing simple games and highlights the potential for creating interactive experiences using command-line interfaces.

Table of Contents

List of Figures

1. Introduction

1.1 Challenges

Limited Graphics and User Interface: Bash scripting is primarily text-based, so creating a graphical user interface (GUI) for the game using utilities like **zenity** can be challenging. The interface might be less visually appealing compared to games developed with dedicated GUI frameworks.

Input Validation: Ensuring that user inputs are valid, such as column selections within the range of the game board or handling non-numeric inputs, requires robust input validation logic.

Game Logic Complexity: While Connect 4 is a relatively simple game, implementing the game logic for dropping pieces, checking for win conditions (including horizontal, vertical, and diagonal), and handling tie conditions requires careful consideration and testing.

Testing and Debugging: Testing the script thoroughly to identify and fix any bugs or unexpected behavior, especially in edge cases, is crucial for ensuring the game functions correctly and reliably.

Graphical Representation: Displaying the grids adjacent to each other and getting the layout of the grid that resembles the actual Connect 4 game was challenging. Finding methods to replace space with the respective user's mark was complicated to resolve.

1.2 Motivation For The Work

Learning and Practice: Developing a game in Bash provides an opportunity to practice and improve scripting skills, including handling data structures, conditional statements, loops, and user input/output.

Fun and Entertainment: Developing and playing games can be a fun and rewarding experience. Creating a Connect 4 game allows developers to enjoy the process of designing and implementing a game while also providing entertainment for themselves and others.

Practical Applications: While Bash scripting is not typically associated with game development, creating a game demonstrates the versatility of the language and its potential for practical applications beyond traditional system administration tasks.

2. Methodology and Framework

2.1 System Requirements

Operating System: The script should run on any Unix-like operating system, including Linux and macOS, which have native support for Bash scripting. Windows users can run the script using a Bash emulator or subsystem such as Cygwin, Windows Subsystem for Linux (WSL), or Git Bash.

Bash Shell: The script relies on features specific to the Bash shell, so a compatible version of Bash (typically included by default on Unix-like systems) is required. For Windows users, a Bash emulator or subsystem as mentioned above is necessary.

zenity Utility: The script utilizes the **zenity** utility for creating graphical dialog boxes. **zenity** is commonly available on Linux distributions with GNOME desktop environments. Windows users may need to install a compatible alternative or adapt the script to use a different method for user input/output.

Graphics Environment: Since the game relies on **zenity** for graphical dialog boxes, a graphical environment is required to run the script. On Linux and macOS systems, this typically means running the script within an X Window System (X11) or similar graphical environment. For Windows users, running the script within a graphical environment provided by a Bash emulator or subsystem is necessary.

2.2 Algorithm & Technique

Game Board Representation: The game board is represented as a two-dimensional array, where each cell corresponds to a position on the grid. This representation allows for efficient manipulation of the game state, including placing pieces and checking for win conditions.

User Input Handling: User input, such as column selections for placing pieces, is obtained using graphical dialog boxes provided by the **zenity** utility. Input validation ensures that only valid column numbers within the range of the game board are accepted.

Game Initialization: The game board is initialized with empty cells represented by dots (.) using a nested loop. This initialization ensures that the game starts with an empty board.

Piece Placement: When a player selects a column to place their piece, the script iterates over the column from bottom to top to find the first empty cell where the piece can be placed. This process ensures that pieces stack on top of each other correctly.

Win Condition Checking: The script checks for win conditions after each player's move to determine if a player has connected four of their pieces horizontally, vertically, or diagonally. This involves iterating over rows, columns, and diagonals to search for sequences of four identical pieces.

Column Full Checking: Before placing a piece in a column, the script checks if the column is full to prevent further pieces from being placed. This involves checking if the top row of the column is occupied by a piece.

Player Alternation: The game alternates between two players (X and O) after each move, ensuring fair gameplay.

Dialog Box Display: The game state, including the current configuration of the game board, is displayed to the players using graphical dialog boxes provided by **zenity**. This allows players to visualize the game board and make informed decisions.

Error Handling: The script includes error handling mechanisms to handle invalid user inputs, such as non-numeric column selections or attempts to place pieces in full columns. Error messages are displayed to the user to prompt them to enter valid inputs.

3. Implementation

3.1 Main UNIX Commands

zenity: A command-line utility that allows for the creation of graphical dialog boxes in shell scripts. It's used for displaying messages and obtaining user input in the Connect 4 game.

declare: A built-in Bash command used to declare variables, including associative arrays. In the script, it's used to declare the game board as an associative array.

for: A loop control structure used for iterating over a range of values. In the script, **for** loops are used for initializing the game board, printing the board, and checking for winning combinations.

if: A conditional control structure used for executing code based on certain conditions. In the script, **if** statements are used for checking if a column is full, validating user input, and determining if there's a winning combination.

return: A command used to return a value from a function. In the script, **return** is used to indicate whether a column is full or if there's a winning combination.

while: A loop control structure used to repeatedly execute code as long as a condition is true. In the script, a **while** loop is used to continue the game until a player wins or the game is exited.

function: A keyword used to define shell functions. In the script, functions are defined for initializing the game board, printing the board, checking if a column is full, dropping a piece into a column, checking for a winning combination, and running the main game loop.

3.2.1 Uses

Interactive Entertainment: The primary use of the project is to provide interactive entertainment by allowing users to play the classic Connect 4 game. Players can enjoy a fun and challenging gaming experience directly from their command-line interface, enhancing their leisure time.

Learning and Skill Development: The project offers a platform for users to learn and develop their skills in Bash scripting. By examining the script's code and experimenting

with it, users can gain a better understanding of Bash programming concepts, such as loops, conditional statements, functions, and array manipulation.

Problem-Solving and Logic: Playing the Connect 4 game requires strategic thinking, problem-solving, and logical reasoning. Users can improve these cognitive skills by analyzing the game board, planning their moves, and anticipating their opponent's actions.

User Interface Interaction: The use of the **zenity** utility for graphical dialog boxes introduces users to interacting with a graphical user interface (GUI) within the command-line environment. Users can experience GUI elements such as input dialogs and message boxes, broadening their understanding of interface design and interaction principles.

3.2.2 Source code

```
initialize_board() {
    for ((i=0; i<rows; i++)); do
        for ((j=0; j<columns; j++)); do
            board[$i,$j]="."
        done
    done
}
# Game board
print_board() {
    local board_str=""
    for ((i=0; i<rows; i++)); do
        for ((j=0; j<columns; j++)); do
            board_str+="| ${board[$i,$j]} "
        done
        board_str+="|\n"
    done
        zenity --info --title="Connect 4" --text="$board_str"
--width=200 --height=150
}
# Function to check if a column is full
column_full() {
```

```
    local col=$1
    if [[ ${board[0,$col]} == "." ]]; then
        return 1
    else
        return 0
    fi
}
# Function to drop a piece into a column
drop_piece() {
    local col=$1
    local symbol=$2
    for ((i=rows-1; i>=0; i--)); do
        if [[ ${board[$i,$col]} == "." ]]; then
            board[$i,$col]=$symbol
            break
        fi
    done
}
# Function to check if there's a winning combination
check_win() {
    local symbol=$1
    # Check horizontally
    for ((i=0; i<rows; i++)); do
        for ((j=0; j<=columns-4; j++)); do
                        if [[ ${board[$i,$j]} == "$symbol" &&
${board[$i,$((j+1))]} == "$symbol" && ${board[$i,$((j+2))]} ==
"$symbol" && ${board[$i,$((j+3))]} == "$symbol" ]]; then
                return 0
            fi
        done
    done
    # Check vertically
    for ((j=0; j<columns; j++)); do
        for ((i=0; i<=rows-4; i++)); do
```

```
                        if [[ ${board[$i,$j]} == "$symbol" &&
${board[$((i+1)),$j]} == "$symbol" && ${board[$((i+2)),$j]} ==
"$symbol" && ${board[$((i+3)),$j]} == "$symbol" ]]; then
                return 0
            fi
        done
    done
    # Check diagonally
    for((i=0; i<=rows-4; i++)); do
        for ((j=0; j<=columns-4; j++)); do
                        if [[ ${board[$i,$j]} == "$symbol" &&
${board[$((i+1)),$((j+1))]}        ==        "$symbol"        &&
${board[$((i+2)),$((j+2))]}        ==        "$symbol"        &&
${board[$((i+3)),$((j+3))]} == "$symbol" ]]; then
                return 0
            fi
        done
    done
    # Check anti-diagonally
    for ((i=0; i<=rows-4; i++)); do
        for ((j=3; j<columns; j++)); do
                        if [[ ${board[$i,$j]} == "$symbol" &&
${board[$((i+1)),$((j-1))]}        ==        "$symbol"        &&
${board[$((i+2)),$((j-2))]}        ==        "$symbol"        &&
${board[$((i+3)),$((j-3))]} == "$symbol" ]]; then
                return 0
            fi
        done
    done
    return 1
}
# Main game function
connect4() {
    rows=6
    columns=7
    declare -A board
```

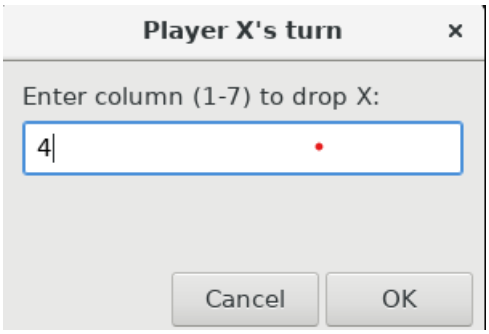
```
    initialize_board
    player="X"

    while true; do
        print_board
          column=$(zenity --entry --title="Player $player's turn" --text="Enter column (1-$columns) to drop $player:" --entry-text "")
        if [[ $column =~ ^[1-7]$ ]]; then
            if column_full $((column-1)); then
                                zenity --info --title="Column Full" --text="Column $column is full! Choose another column." --width=200
            else
                drop_piece $((column-1)) $player
                if check_win $player; then
                    print_board
                                    zenity --info --title="Game Over" --text="Player $player wins!" --width=200
                    break
                fi
                if [[ $player == "X" ]]; then
                    player="O"
                else
                    player="X"
                fi
            fi
        else
                            zenity --info --title="Invalid Column" --text="Invalid column! Please enter a number between 1 and $columns." --width=200
        fi
    done
}
# Start the game
connect4
```

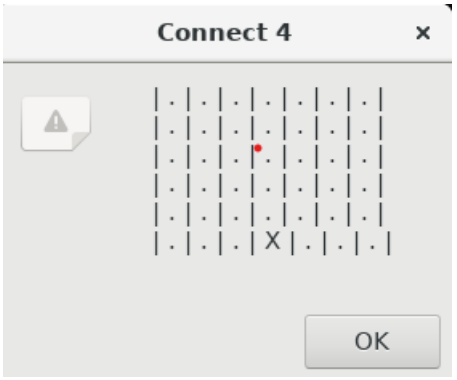
4. Result and Analysis

4.1 Output of the Script

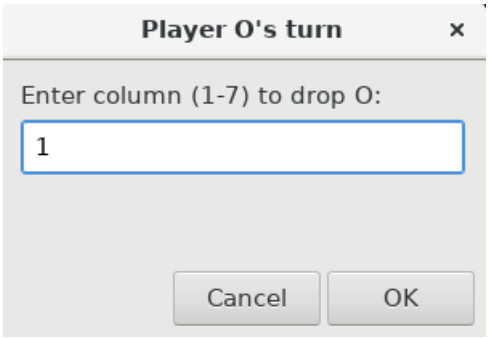
4.1.1 Output with dialogue boxes using zenity utility



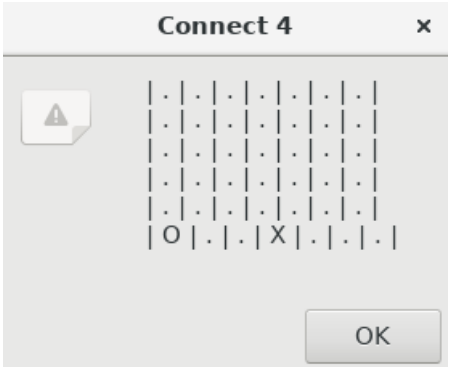
Player X choose column 4



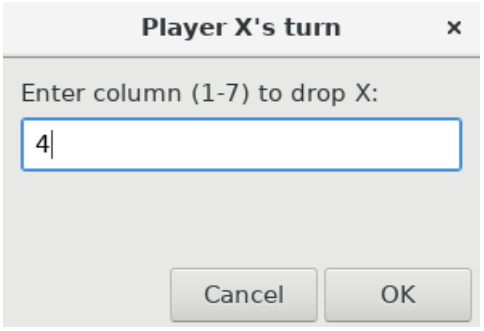
Output

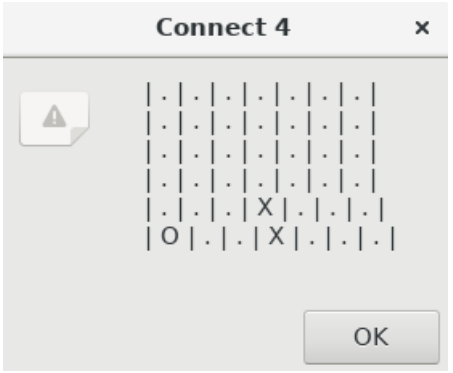


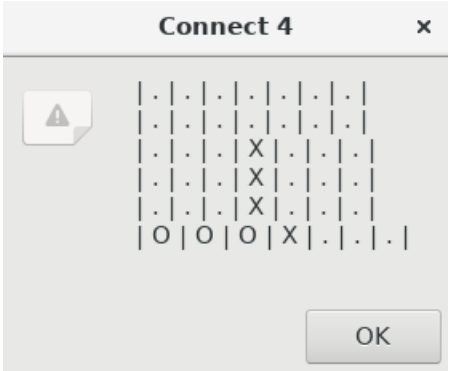
Player O chooses column 1



Output



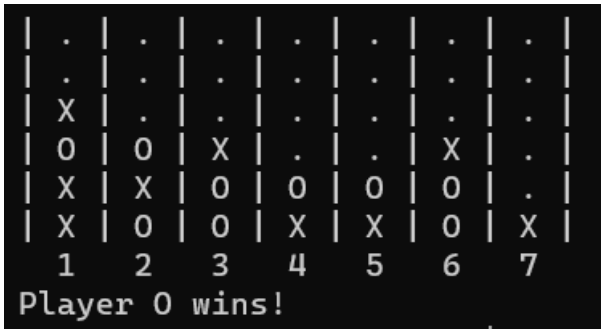




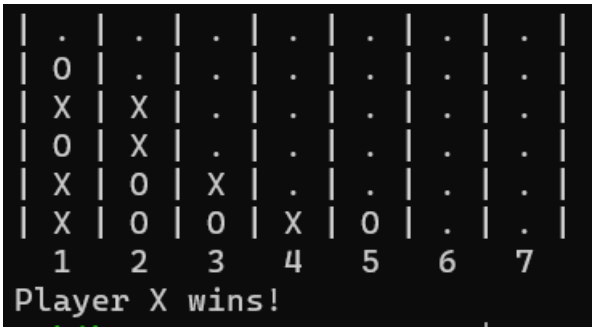
Player X drops the 4th piece

Player X wins

4.1.2 Output using Ubuntu Terminal

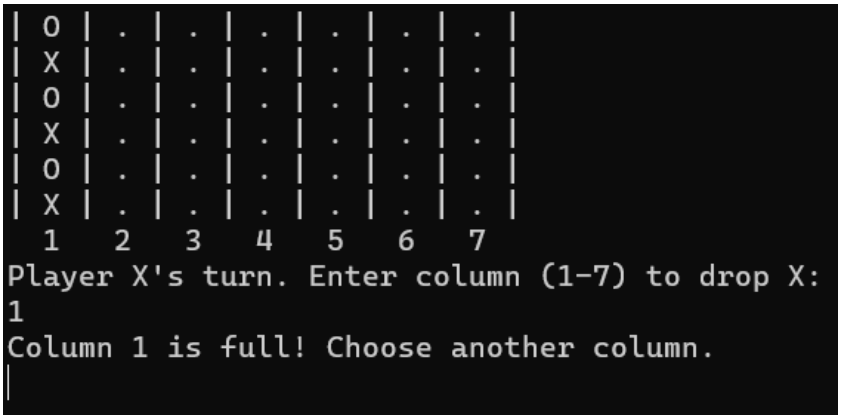


Horizontal Win

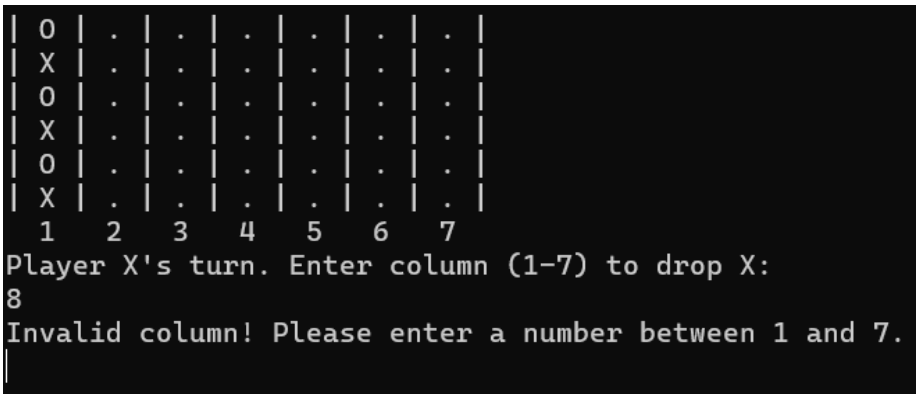


Diagonal win

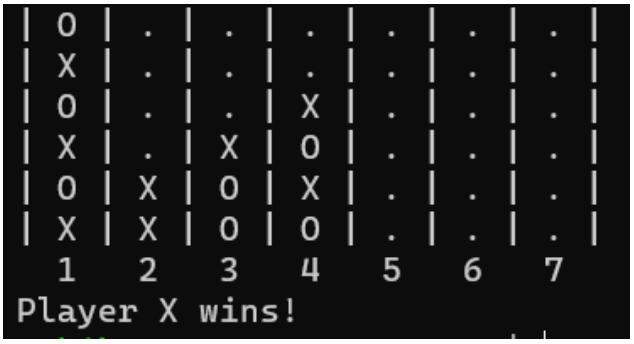
Horizontal win



Column full display

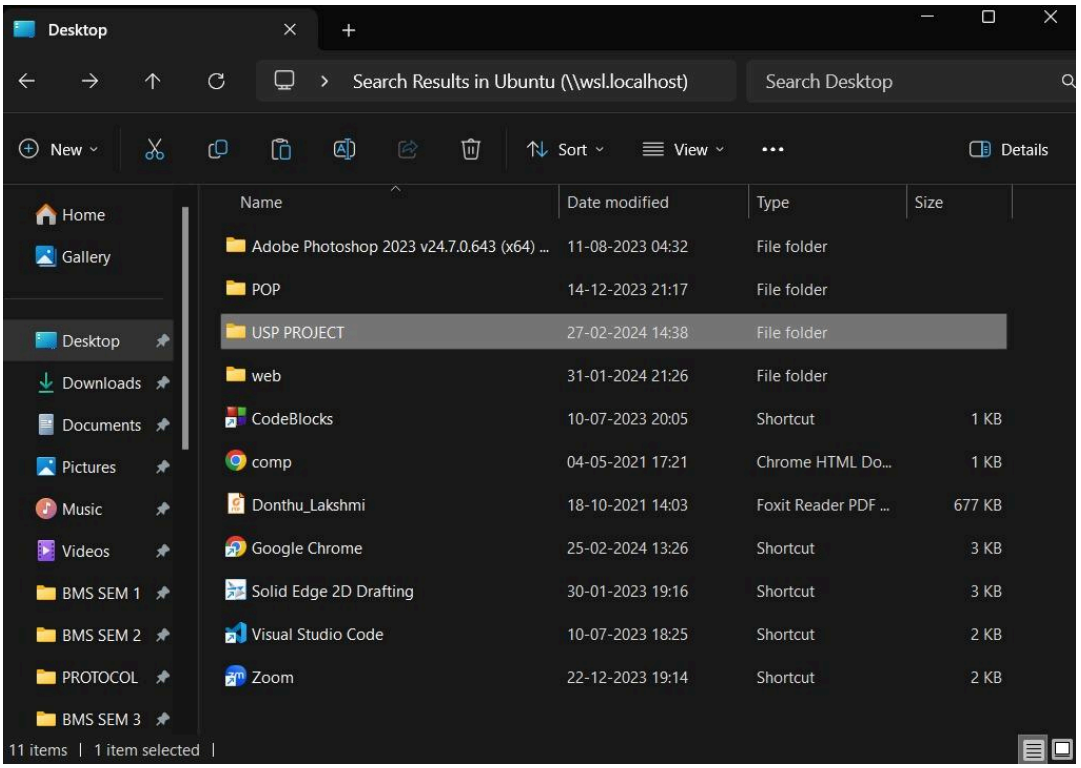


Column input out of bounds

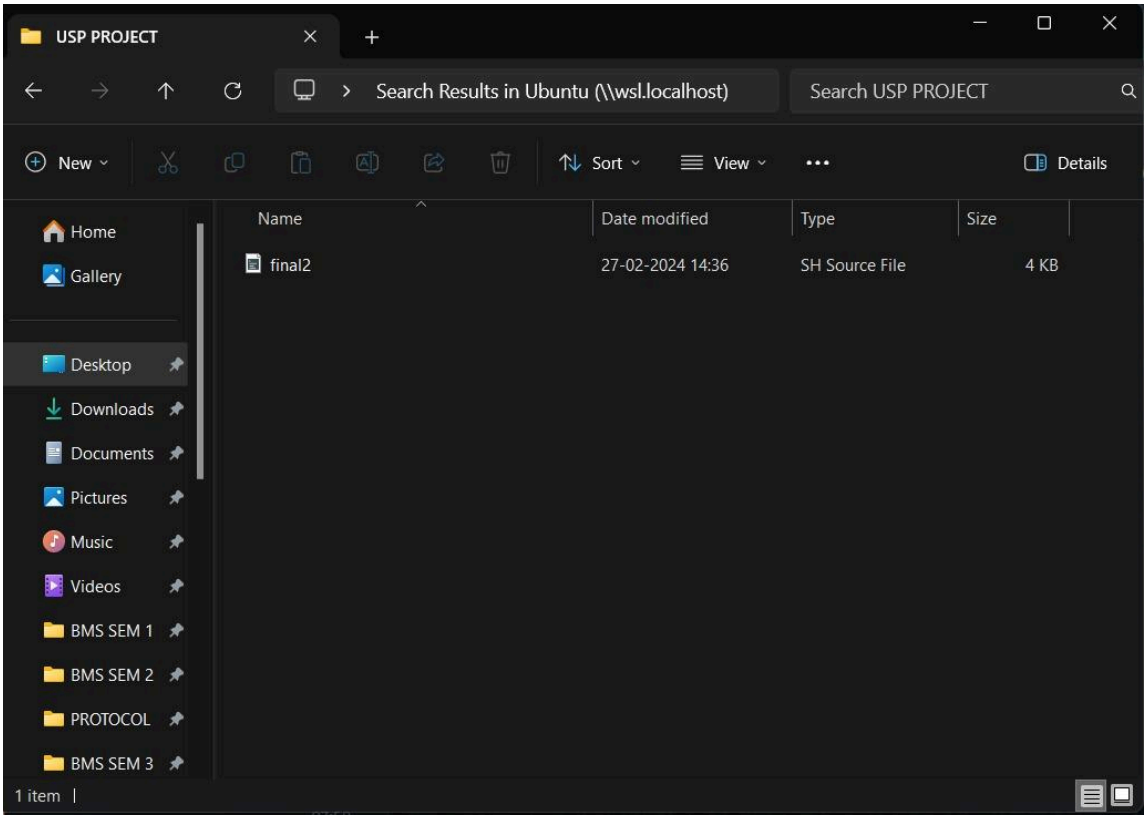


Anti diagonal win

4.2 Downloaded Folders



4.3 Downloaded Files



5. Conclusion

In conclusion, the implementation of the Connect 4 game in Bash scripting offers a versatile and engaging experience within the command-line environment. Through the utilization of fundamental programming concepts and the zenity utility for graphical dialog boxes, this project demonstrates the potential of Bash scripting beyond its traditional use in system administration tasks.

The Connect 4 game serves as an entertaining pastime, challenging players to employ strategic thinking and logical reasoning while enjoying a classic board game experience. Additionally, the project provides valuable learning opportunities for users interested in Bash scripting, enabling them to explore concepts such as loops, conditional statements, functions, and array manipulation in a practical context.

Furthermore, the integration of graphical user interface elements using zenity enhances the user experience by providing intuitive interactions and visual feedback, bridging the gap between command-line interfaces and graphical environments.

Overall, the Connect 4 game in Bash scripting exemplifies the creativity, versatility, and educational value inherent in script-based programming. Whether used for entertainment, learning, or community engagement, this project underscores the potential of Bash scripting as a tool for innovation and exploration in various domains.

References

[1]N. Pandit, “Introduction to UNIX system,” *GeeksforGeeks*, Aug. 31, 2018. https://www.geeksforgeeks.org/introduction-to-unix-system/

[2]“UNIX / LINUX Tutorial - Tutorialspoint,” *www.tutorialspoint.com*. https://www.tutorialspoint.com/unix/index.htm

[3]“UNIX operating system - javatpoint,” *www.javatpoint.com*. https://www.javatpoint.com/unix-operating-system

[4]P. Singh, “Mastering the Fundamentals of Using Zenity Linux,” *Analytics Vidhya*, Feb. 06, 2023. https://www.analyticsvidhya.com/blog/2023/02/mastering-the-fundamentals-of-using-zenity-linux/

[5]“How to use Zenity with examples,” *www.youtube.com*. https://www.youtube.com/watch?v=TQQxKXQq_t0

[6]“Zenity Manual,” *Gnome.org*, 2014

https://help.gnome.org/users/zenity/stable

[7]Sumitabha Das, *UNIX concepts and applications*. New Delhi: Tata Mcgraw-Hill, 2006.