**MINESWEEPER GAME**

A

SPPS Project Report

*Submitted in partial fulfilment of the*

*Requirements for the award of the Degree of*

**BACHELOR OF ENGINEERING**

IN

**INFORMATION TECHNOLOGY**

BY

Sai Vallabha Linga (1602-24-737-149)

Lokesh Mantoliya (1602-24-737-150)



**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**ACCREDITED BY NAAC WITH ‘A++’ GRADE**

**(Affiliated to Osmania University and Approved by AICTE)**

**Ibrahimbagh, Hyderabad-31**

**2025**

**Vasavi College of Engineering (Autonomous)**

**ACCREDITED BY NAAC WITH ‘A++’ GRADE**

**(Affiliated to Osmania University and Approved by AICTE)**

**Hyderabad-500 031**

**Department of Information Technology**



**DECLARATION BY THE CANDIDATE**

We, **Sai Vallabha Linga** and **Lokesh Mantoliya**, bearing hall ticket numbers, **1602-24-737-149**and **1602-24-737-150**, hereby declare that the project report entitled “**MINESWEEPER GAME”** is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology.**

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**Sai Vallabha Linga**

**1602-24-737-149**

**Lokesh Mantoliya**

**1602-24-737-150**

(Faculty in charge) (Head, Dept of IT)

**ACKNOWLEDGMENT**

We extend our sincere thanks to **Dr. S. V. Ramana, Principal**, Vasavi College of Engineering for his encouragement.

We express our sincere gratitude to **Dr. K. Ram Mohan Rao, Professor & Head**, Department of Information Technology, Vasavi College of Engineering, for introducing the Mini-Project module in our curriculum, and also for his suggestions, motivation, and co-operation for the successful completion of our Mini Project.

We also want to thank and convey our gratitude towards our **SPPS Faculty, Sruthi Anand** for guiding us in understanding the process of project development & giving us timely suggestions at every phase.

**Sai Vallabha Linga**

**1602-24-737-149**

**Lokesh Mantoliya**

**1602-24-737-149**

**ABSTRACT**

This project presents a C-based implementation of the classic logic game *Minesweeper*, where players aim to clear a rectangular grid without triggering any hidden mines. Each cell in the grid may either contain a mine or a number indicating how many adjacent cells contain mines. The game begins with a blank grid, and as the player reveals cells, they must use logic and deduction to avoid uncovering mines. A single misstep can end the game, adding a layer of tension and strategy to each move. The game dynamically generates the minefield and responds to user inputs in real time, offering an engaging and challenging experience.

The development of this game involved applying key programming concepts such as two-dimensional arrays, recursion (for automatic revealing of empty cells), and input validation. The project emphasizes the importance of user interaction, efficient memory management, and algorithmic thinking. Beyond serving as a fun and nostalgic recreation of a well-known game, this implementation demonstrates how fundamental C programming techniques can be leveraged to build interactive, logic-based applications. Additional features such as customizable grid size and mine count can enhance the game's replayability and complexity.

**INTRODUCTION**

**Domain Description**

* The project is a text-based implementation of the classic Minesweeper game.
* It belongs to the game development domain, focused on logical reasoning, problem-solving, and user interaction through the console.
* This game demonstrates data structure usage (2D arrays), random number generation and recursive algorithms.

### **Features ot he project**

**1. Game Board Creation**

* A 13x13 grid is created using 2D arrays.
* One grid (board[][]) stores the **actual game logic (mines and numbers)**.
* Another grid (display[][]) represents the **player’s view**, hiding unrevealed cells.

**2. Random Mine Placement**

* Places **20 mines** randomly on the board using rand().
* Ensures no duplicate placement and adjusts neighbouring cell values to indicate adjacent mine count ('1' to '8').

**3. Dynamic Gameplay Loop**

* The game runs in a **loop**, repeatedly:
  + Printing the visible grid.
  + Taking user input for a move.
  + Revealing the chosen cell.
  + Checking for win/loss conditions.

**4. Play Time Tracking**

* Uses clock() to measure how long the user took to play the game.

**5. Recursive Cell Reveal Logic**

* If the player reveals a cell with 0 adjacent mines, the program automatically **reveals all surrounding safe cells recursively**.

**6. Win/Loss Detection**

* The game ends in two scenarios:
  + **Loss**: Player selects a mine cell.
  + **Win**: All non-mine cells are revealed.
* Shows the full board at the end for review.

**7. Console Interface**

* All interactions happen through a terminal window.
* Clear formatting for row/column headers and cells to make the interface user-friendly.

**TECHNOLOGY**

### **Software Requirements**

* Windows 10/11 OR macOS 10.13 or later
* Code Editing and Debugging - Code::Blocks
* Visual Studio Code (VS Code)
* Compiler - GCC (GNU Compiler Collection)
* Terminal - Windows Terminal
* Documentation Tools - MS Word
* To host code online – GitHub

### **Hardware Requirements**

* PC/Laptop with a minimum of 4GB RAM
* Processor : Dual-core or higher.
* Minimum disk space : 500MB Hardware Requirements

### 

### 

### **PROGRAM SCRIPT**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 5 //max size=16

#define MINES 3

char board[SIZE][SIZE]; //actual board

char display[SIZE][SIZE]; //board which is displayed

void initBoard()

{

int i, j;

for (i = 0; i < SIZE; i++)

{

for (j = 0; j < SIZE; j++)

{

board[i][j] = '0'; //initializes every tile to '0'

display[i][j] = '-'; //initialises to every tile '-'

}

}

}

void placeMines() //place mines randomly

{

int count = 0;

while (count < MINES)

{

int x = rand() % SIZE;

int y = rand() % SIZE;

if (board[x][y] != '\*')

{

board[x][y] = '\*';

count++;

// Update count around the mine

for (int i = -1; i <= 1; i++)

{

for (int j = -1; j <= 1; j++)

{

int nx = x + i;

int ny = y + j;

if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE && board[nx][ny] != '\*')

{

board[nx][ny]++; //update count by 1

}

}

}

}

}}

void printBoard(char b[SIZE][SIZE]) //to print board

{

printf(" ");

for (int i = 0; i < SIZE; i++)

{

printf("%02d ", i);

}

printf("\n");

printf("\n");

for (int i = 0; i < SIZE; i++)

{

printf("%02d ", i);

for (int j = 0; j < SIZE; j++)

{

printf("%2c ", b[i][j]);

}

printf("\n");

}

}

int reveal(int x, int y) //reveals the entered coordinates

{

if (x < 0 || x >= SIZE || y < 0 || y >= SIZE || display[x][y] != '-')

return 0;

display[x][y] = board[x][y];

if (board[x][y] == '\*')

return -1; // Hit a mine

if (board[x][y] == '0')

{

for (int i = -1; i <= 1; i++)

{

for (int j = -1; j <= 1; j++)

{

if (i != 0 || j != 0)

reveal(x + i, y + j); //recursive call

}

}

}

return 1;

}

int checkWin() //checks win or not

{

int revealed = 0;

for (int i = 0; i < SIZE; i++)

{

for (int j = 0; j < SIZE; j++)

{

if (display[i][j] != '-')

revealed++;

}

}

return (revealed == SIZE \* SIZE - MINES);

}

int main()

{

printf("=====================GAME STARTS=====================\n\n");

int x, y;

srand(time(NULL)); //seeds a random according to time

initBoard();

placeMines();

clock\_t start, end;

double play\_time;

start = clock(); //start timer

while(1)

{

printBoard(display);

printf("\nEnter coordinates to reveal (row col) : ");

scanf("%d %d", &x, &y);

if (x >= SIZE || y >= SIZE)

printf("\nPlease enter valid coordinates !!\n");

printf("\n----------------------------------------------------\n\n");

int result = reveal(x, y);

if (result == -1)

{

printBoard(board);

printf("\nGame Over! You hit a mine.\n\n");

break;

}

else if (checkWin())

{

printBoard(board);

printf("\nCongratulations! You cleared the board.\n\n");

break;

}

}

end = clock(); //end timer

play\_time = ((double)(end - start)) / CLOCKS\_PER\_SEC; //calculation of playtime

printf("Time taken : %lf seconds\n", play\_time);

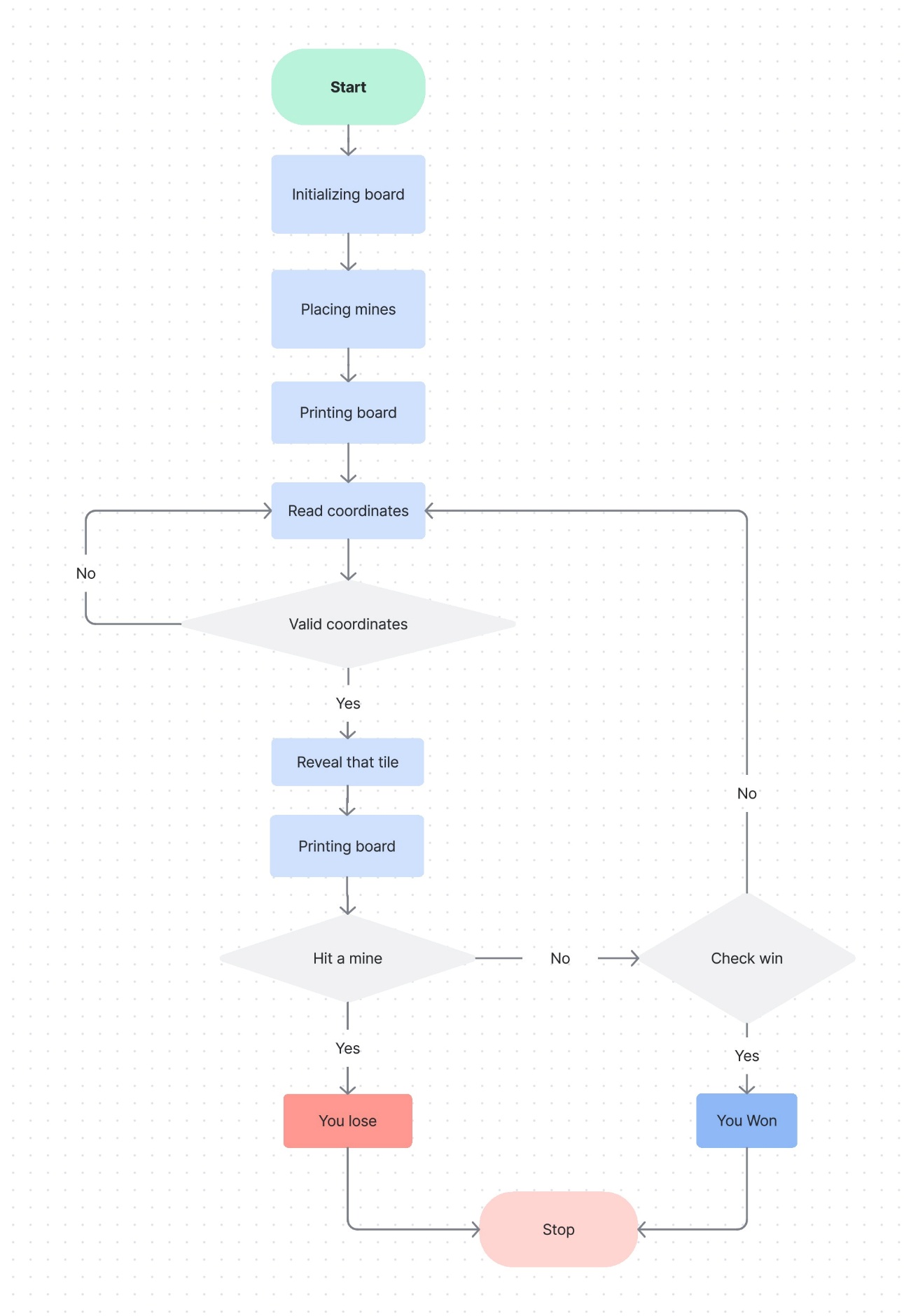
printf("\n=====================================================\n");

return 0;

}

**PROPOSED WORK**

Here is a flow Chart representing the flow of game code.



**MODULE-WISE CODE**

**Global Constants and Variables**

* #define SIZE n  
  Sets the board size (n×n).
* #define MINES num  
  Total number of mines on the board.
* char board[SIZE][SIZE]  
  Stores the actual board with mines and adjacent mine counts.
* char display[SIZE][SIZE]  
  Stores what the user sees ('-' for unrevealed, numbers for revealed).

**Module: Board Initialization**

void initBoard()

* Initializes both board and display arrays.
* Fills
  + board[][] with '0' (indicating no mines around).
  + display[][] with '-' (indicating all cells are unrevealed).

**Module: Mine Placement and Mine Count Update**

void placeMines()

* Randomly places MINES (num) '\*' on the board.
* After placing a mine, it
  + Increments surrounding cells' values (board[i][j]++) to count nearby mines.
  + Only increments non-mine cells.
  + This way, '1' to '8' indicate how many mines surround a cell.

**Module: Display**

void printBoard(char b[SIZE][SIZE])

* Prints the board (either board or display) in a readable grid format.
* Shows row and column numbers with formatted output.

**Module: Game Logic**

int reveal(int x, int y)

* Reveals a cell at coordinates (x, y).
* **Returns:**
  + -1 if a mine is hit.
  + 1 for a successful reveal.
  + 0 if coordinates are invalid or already revealed.
* If the revealed cell is '0' (no adjacent mines), it **recursively reveals** its neighbors.

int checkWin()

* Counts all revealed cells.
* **Returns 1 (true)** if total revealed cells = total safe cells (SIZE × SIZE - MINES), **else 0**.

**Module: Main Game Loop**

int main()

* Displays game start message.
* Calls initBoard() and placeMines().
* Starts a timer to measure play time.
* Enters a loop:
  + Prints the current visible board (display).
  + Takes user input (x y) to reveal a cell.
  + Checks if the user hit a mine → ends game with loss.
  + Checks if all safe cells are revealed → ends game with win.
* Ends the timer and prints the total time taken.

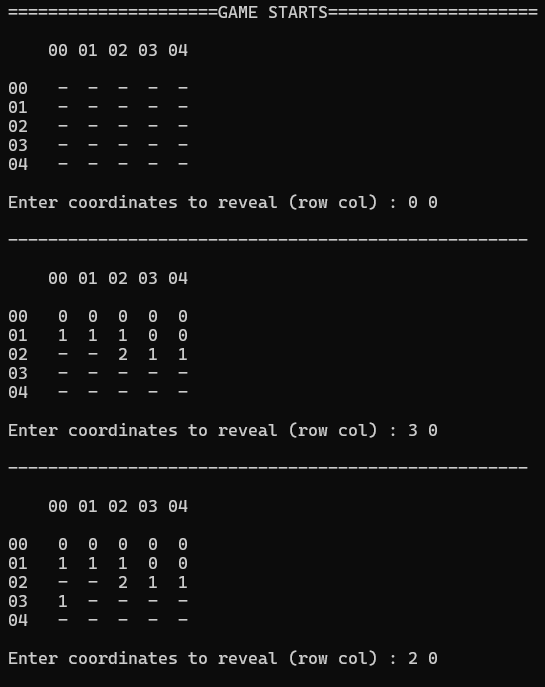
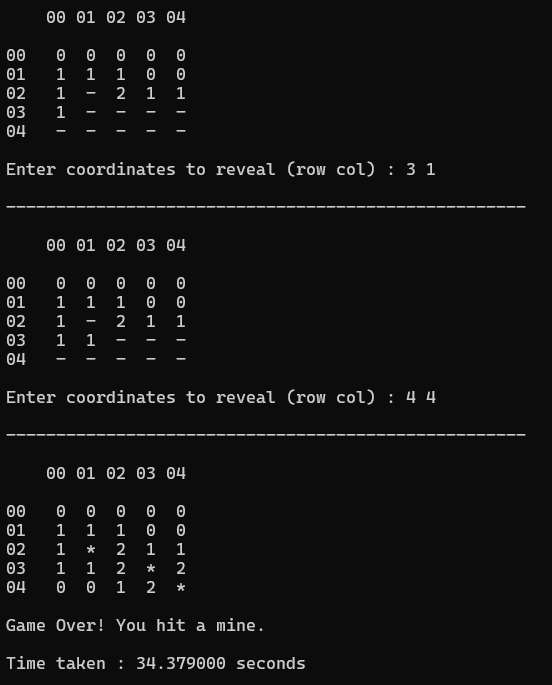
**Timer Functionality**

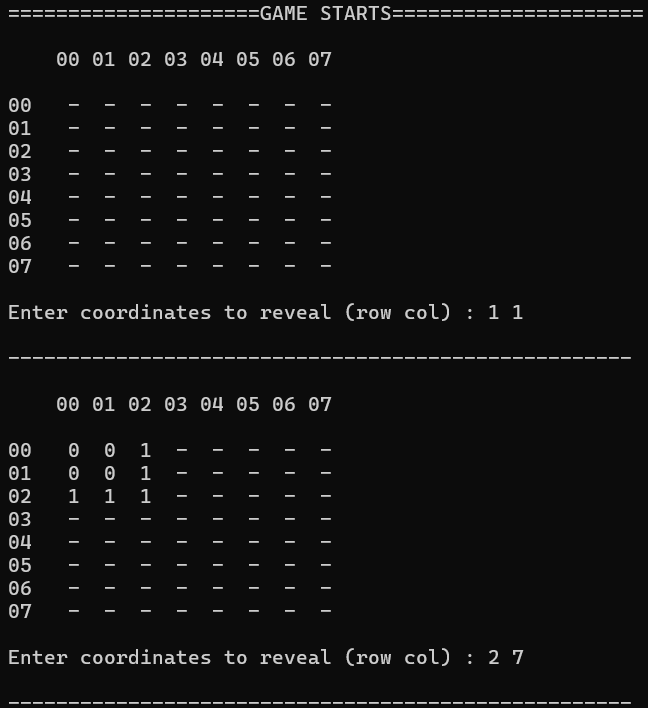
* Uses clock() to record time at game start and end.
* Computes and prints total play time in seconds.

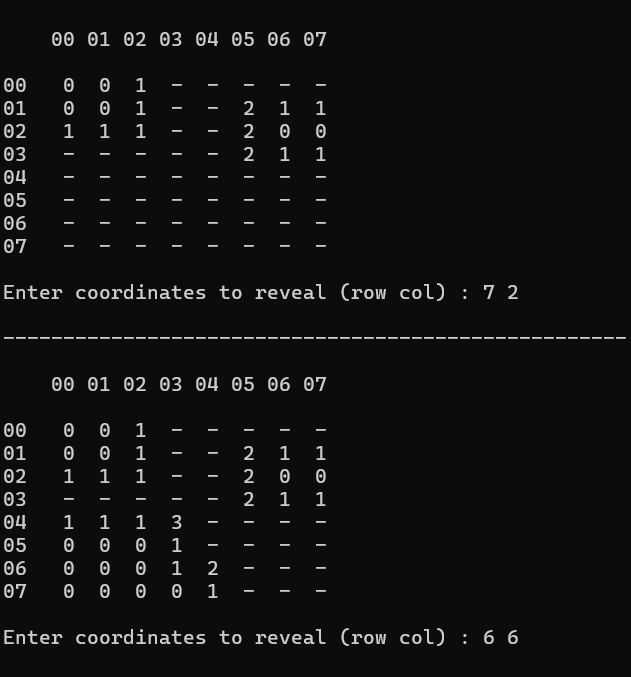
**RESULTS**

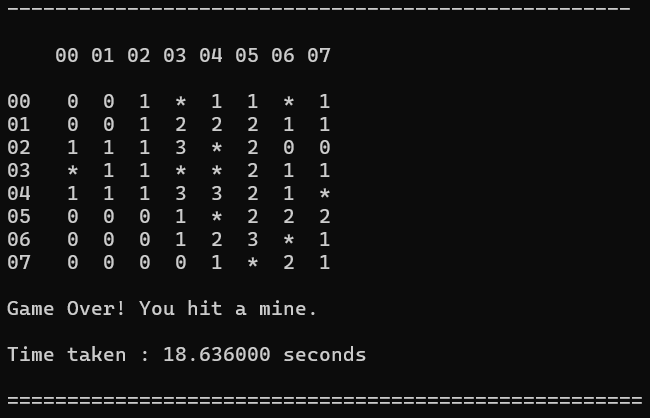
(Outputs)

**For 5\*5 grid**

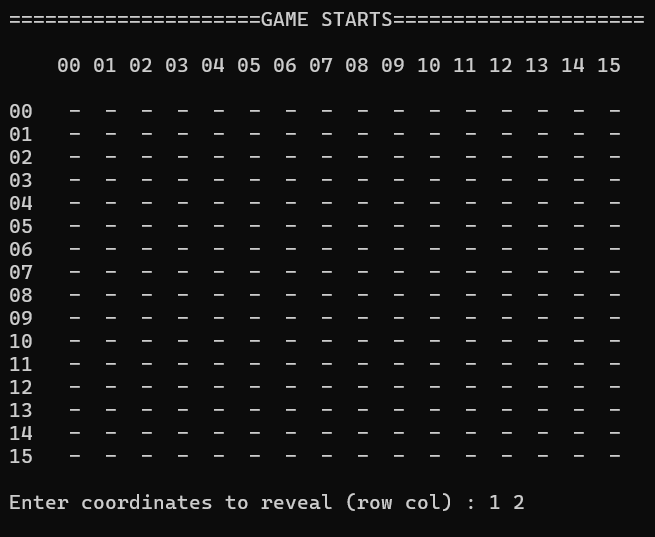


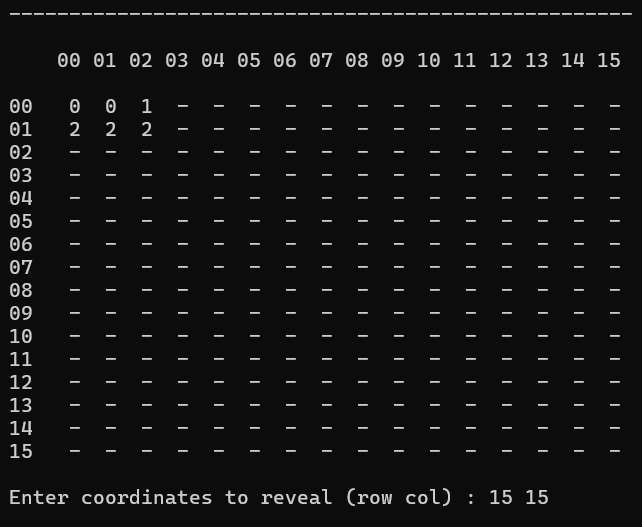
**For 8\*8 grid**

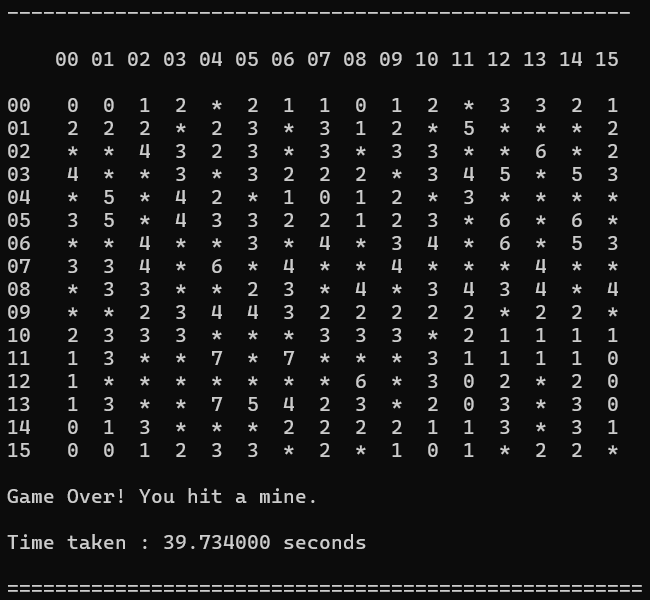




**For 16\*16 grid**

****

****



**REFERENCES**

* Original Game Concept

Microsoft. *Minesweeper Game*. [Windows Entertainment Pack]. Microsoft Corporation, 1990.

* Game Rules and Mechanics

<https://en.wikipedia.org/wiki/Minesweeper_(video_game)>

* Random Number Generation (for Mines Placement)

<https://www.geeksforgeeks.org/c-rand-function/>

<https://www.w3schools.com/c/ref_stdlib_srand.php>(For srand() function)

* For Board Set Up (2D Arrays)

<https://www.geeksforgeeks.org/multidimensional-arrays-c-cpp/>

* For Designing Flowchart

<https://www.atlassian.com>

**CONCLUSION**

The **Minesweeper Game** developed in the C programming language successfully replicates the core functionality of the classic puzzle game in a console-based environment. This project demonstrates effective use of fundamental programming concepts such as **2D arrays, recursion, random number generation, input validation**, and **time measurement**.

Through this project, key problem-solving and logical thinking skills were reinforced, particularly in handling recursive cell revealing and win/loss conditions. The program offers a user-friendly experience via a clear text-based interface and ensures smooth gameplay with proper validations and feedback.

Overall, the project not only strengthens understanding of C programming but also provides a solid foundation for developing more advanced games or transitioning this logic into a **graphical interface** in the future. It stands as a complete, functional, and engaging game that highlights the potential of console-based application development.