



**CMR UNIVERSITY**

Private University Established in Karnataka State by Act No. 45 of 2013

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

**Project Work**

*On*

**“Tic Tac Toe”**

For the requirement of 1<sup>th</sup> Semester (4CSPL101 – Problem Solving Using Python) B.Tech. in  
Computer Science and Engineering

***SUBMITTED BY***

**KAVYA SHREE- (24BBTCA060)**

**POOJA RAJPUROHIT- (24BBTCA084)**

**NEYSA MARY PRAMOD- (24BBTCA0078)**

**S ADITI REDDY- (24BBTCA099)**

*Submitted to*

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CMR University**

Off Hennur - Bagalur Main Road,

Near Kempegowda International Airport, Chagalahatti,

Bengaluru, Karnataka-562149

Academic Year - 2024-25



**CMR UNIVERSITY**

Private University Established in Karnataka State by Act No. 45 of 2013

## SCHOOL OF ENGINEERING AND TECHNOLOGY

Chagalahatti, Bengaluru, Karnataka-562149

### Department of Computer Science and Engineering

## CERTIFICATE

Certified that the Project Work entitled "**Tic Tac Toe**" carried out by **Kavya Shree V(24BBTCA060), Pooja Rajpurohit (24BBTCA084), Neysa Mary Pramod (24BBTCA078) and S Aditi Reddy (24BBTCA099)** Bonafide students of **SCHOOL OF ENGINEERING AND TECHNOLOGY**, in partial fulfillment for the award of **BACHELOR OF TECHNOLOGY** in 1<sup>th</sup> Semester Computer Science and Engineering of **CMR UNIVERSITY**, Bengaluru during the year 2025. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report. The project has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

#### Course in charge

**(Dr. Gyanappa A Walikar)**

Assistant Professor

Dept. of CSE

SOET, CMRU, Bengaluru.

**H.O. D**

**(Dr. S P Manikandan)**

Head of Department (CST/CE)

SOET, CMRU, Bengaluru.

# **ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of this project would be incomplete without the mention of the people who made it possible, without whose constant guidance and encouragement would have made efforts go in vain.

We would like to express our thanks to **Dr. S P Manikandan**, Head of Department **Computer Science and Technology**, CMR University, Bengaluru, for **her** encouragement that motivated us for the successful completion of Project work.

We express our thanks to our Internal Project Guide **Dr. Gyanappa A Walikar**, **Assistant Professor**, Department of Computer Science and Engineering, School of Engineering and Technology, CMR University for her constant support.

**Kavya Shree V(24BBTCA060)**  
**Pooja Rajporith (24BBTCA084)**  
**Neysa Mary Pramod (24BBTCA078)**  
**S Aditi Reddy (24BBTCA099)**

I Kavya Shree V(24BBTCA060), Pooja Rajpurohit(24BBTCA084),  
Neysa Mary Pramod(24BBTCA078) and S Aditi Reddy (24BBTCA099), students of  
**1<sup>st</sup> semester B.Tech, Computer Science and Technology**, School of Engineering and  
Technology, Bengaluru, hereby declare that the project work entitled "**Tic Tac Toe**"  
has been carried out by us under the guidance of **Dr. Gyanappa A Walikar**, Assistant  
Professor, Department of Computer Science and Engineering, School of Engineering and  
Technology. This report is submitted in partial fulfillment of the requirement for award of  
**Bachelor of Technology in Computer Science and Technology**, by CMR University,  
Bengaluru during the academic year 2024-2025. The project report has been approved  
as it satisfies the academic requirements in respect of project work prescribed for the said  
degree.

**Kavya Shree V(24BBTCA060)**  
**Pooja Rajpurohit(24BBTCA084)**  
**Neysa Mary Pramod (24BBTCA078)**  
**S Aditi Reddy (24BBTCA099)**

## **Abstract:**

Tic Tac Toe is a simple yet profound game that has served as a foundation for exploring programming concepts and artificial intelligence. This program implements a digital version of the classic two-player game, offering an interactive and user-friendly interface. The core functionality includes a 3x3 grid where players alternate turns to place their respective markers (e.g., "X" and "O") with the goal of aligning three markers horizontally, vertically, or diagonally. The program can be extended to include a single-player mode featuring an AI opponent. The AI is typically implemented using algorithms such as the Minimax Algorithm, ensuring optimal gameplay by evaluating all possible moves. This highlights the potential for machine intelligence in decision-making processes within finite game spaces.

# CONTENT

SR. NO	TOPIC	PAGE. NO
1	Tic Tac Toe	7
2	Thinker Programming	11
3	Visual Studio Code	16
4	Source Code	17
5	Output	20
6	Conclusion	21

# Tic Tac Toe

## Introduction to Tic-Tac-Toe in Python

Tic-Tac-Toe is a classic two-player game that has been popular for decades. It is a simple yet strategic game played on a 3x3 grid, where two players take turns marking spaces with "X" and "O". The objective is to form a straight line of three identical marks either horizontally, vertically, or diagonally. If all spaces are filled and no player has formed a winning line, the game ends in a draw.

## Why Implement Tic-Tac-Toe in Python?

Tic-Tac-Toe is an excellent beginner-friendly project for learning Python because it involves fundamental programming concepts, such as:

- **Lists and Arrays** – Representing the game board.
- **Loops** – Handling player turns and continuously updating the game state.
- **Conditional Statements** – Checking for a win, loss, or draw.
- **Functions** – Breaking down the game into manageable parts like displaying the board, validating input, and checking for winners.
- **User Input Handling** – Ensuring players enter valid moves.

## Basic Implementation in Python

A simple Python implementation of Tic-Tac-Toe can be done using the command line, where players enter their moves as coordinates. The board is typically represented using a 3x3 list, and the game runs in a loop until a player wins or the board is full.

The program requires functions for:

1. **Displaying the board**
2. **Handling player input**
3. **Checking for a winner**
4. **Detecting a draw**
5. **Switching between players**

## Advanced Versions

For those looking to expand their project, enhancements can include:

- **A Graphical User Interface (GUI):** Using Tkinter or Pygame for a more interactive experience.
- **An AI Opponent:** Implementing the Minimax algorithm to create an unbeatable computer opponent.
- **Multiplayer Mode:** Allowing two players to play online or via a local network.

Tic-Tac-Toe is not just a fun game to play but also a great way to strengthen problem-solving and Python programming skills. Whether you create a basic text-based version or a more advanced AI-powered one, this project serves as an excellent introduction to game development in Python.

Certainly! Here's a brief introduction to the Tic-Tac-Toe game in Python:

Tic-Tac-Toe is a classic two-player game where players take turns marking spaces in a 3x3 grid. The objective is to be the first player to place three of their marks in a horizontal, vertical, or diagonal row. In this Python implementation, we:

1. **Initialize the Board:** Create a 3x3 grid filled with empty spaces.
2. **Print the Board:** Display the current state of the game board.
3. **Player Moves:** Allow players to input their moves and update the board accordingly.
4. **Check for a Winner:** After each move, check if the current player has won the game.
5. **Switch Players:** Alternate between players "X" and "O".
6. **End the Game:** Declare the winner or a tie if the board is full.

This simple implementation focuses on the core mechanics and can be extended with additional features like a graphical user interface (GUI) using libraries like Tkinter or Pygame for a more interactive experience. Happy coding!

# Code Breakdown

## Code Breakdown

### 1. Initialization (`__init__` method)

- Creates the **main window** (`self.master`) with a title and background color.
  - Initializes game variables:
    - `self.current_player`: Tracks whose turn it is (starts with 'X').
    - `self.board`: A list of 9 elements representing the game board (" for empty spots).
    - `self.buttons`: Stores the **Tkinter buttons** for the grid.
  - Creates a **3x3 grid** of buttons inside a Tkinter Frame, and each button calls `make_move(row, col)` when clicked.
  - Displays a **status label** to show whose turn it is.
  - Adds a "**New Game**" **button** to reset the board.
- 

### 2. Player Move (`make_move` method)

- Converts the (row, col) position into a **1D index** (0-8).
  - Checks if the clicked cell is empty:
    - Updates the board with 'X' and updates the button's text.
    - Checks for a **winner**:
      - If **Player X wins**, show a message and reset the game.
      - If the board is full, declare a **tie**.
    - If no winner, switch to **computer's turn** after 500ms (`self.master.after(500, self.computer_move)`).
- 

### 3. Computer Move (`computer_move` method)

- Finds available moves and selects the **best move** using `get_best_move()`.
  - Updates the board with 'O', updates the button, and checks if the **computer wins**.
  - If no winner, switches back to **Player X**.
- 

### 4. AI Logic (`get_best_move` method)

The AI follows a **basic strategy**:

1. **Check if it can win in the next move.**
  2. **Block the player's winning move.**
  3. **Take the center (if available).**
  4. **Take a corner (if available).**
  5. **Pick a random available move.**
- 

### 5. Check Winner (`check_winner` method)

- Defines **winning combinations** (rows, columns, diagonals).
- Checks if any combination has the **same non-empty value**.
- If found, the game ends.

## 6. Highlight Winning Combination (`highlight_winning_combination` method)

- Changes the background color (#90EE90 - light green) of winning cells.

---

## 7. Reset Game (`reset_game` method)

- Clears the board, resets the buttons, and sets the status back to **Player X's turn**.

---

## 8. Running the Game

- `root = tk.Tk()` initializes the main application.
- `game = TicTacToe(root)` creates an instance of the game.
- `root.mainloop()` starts the Tkinter event loop.

---

## Summary

- Player vs. Computer Tic-Tac-Toe
- Graphical UI with Tkinter
- Basic AI using strategic moves
- Winning detection & UI updates
- Reset functionality

## 1. Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

### Example

```
#!/usr/bin/python

import Tkinter top =
Tkinter.Tk()
# Code to add widgets will go here... top.mainloop()
```

This would create a following window –



### Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table –

## Operations and description

These are the operations used in the program

### 1. Arithmetic Operators

Used for mathematical operations.

Operator	Description	Example (a = 10, b = 3)
+	Addition	$a + b \rightarrow 13$
-	Subtraction	$a - b \rightarrow 7$
*	Multiplication	$a * b \rightarrow 30$
/	Division (float)	$a / b \rightarrow 3.3333$
//	Floor Division	$a // b \rightarrow 3$ (removes decimal)
%	Modulus (remainder)	$a \% b \rightarrow 1$
**	Exponentiation	$a ** b \rightarrow 1000 (10^3)$

---

### 2. Comparison (Relational) Operators

Used to compare values, returning True or False.

Operator	Description	Example (a = 5, b = 10)
==	Equal to	$a == b \rightarrow \text{False}$
!=	Not equal to	$a != b \rightarrow \text{True}$
>	Greater than	$a > b \rightarrow \text{False}$
<	Less than	$a < b \rightarrow \text{True}$
>=	Greater or equal	$a >= b \rightarrow \text{False}$
<=	Less or equal	$a <= b \rightarrow \text{True}$

---

### 3. Logical Operators

Used to combine conditional statements.

<b>Operator Description</b>		<b>Example (x = True, y = False)</b>
and	Returns True if both conditions are True	x and y → False
or	Returns True if at least one condition is True	x or y → True
not	Reverses the Boolean value	not x → False

---

## 4. Bitwise Operators

Used to perform bit-level operations.

<b>Operator</b>	<b>Description</b>	<b>Example (a = 5 (0101), b = 3 (0011))</b>
& (AND)	Bitwise AND	a & b → 1 (0001)
~	~ (OR)	Bitwise OR
^ (XOR)	Bitwise XOR	a ^ b → 6 (0110)
~ (NOT)	Bitwise NOT	~a → -6 (Inverts bits)
<< (Left Shift)	Shifts bits left	a << 1 → 10 (1010)
>> (Right Shift)	Shifts bits right	a >> 1 → 2 (0010)

---

## 5. Assignment Operators

Used to assign values to variables.

<b>Operator Description</b>		<b>Example (a = 10)</b>
=	Assign value	a = 10
+=	Add and assign	a += 2 → a = 12
-=	Subtract and assign	a -= 2 → a = 8
*=	Multiply and assign	a *= 2 → a = 20
/=	Divide and assign	a /= 2 → a = 5.0
//=	Floor divide and assign	a // 3 → a = 3
%=	Modulus and assign	a %= 3 → a = 1

---

<b>Operator Description</b>	<b>Example (a = 10)</b>
-----------------------------	-------------------------

<code>**=</code>	Exponentiate and assign a <code>**= 2</code> → a = 100
------------------	--

---

## 6. Identity Operators

Used to compare memory locations of two objects.

<b>Operator Description</b>	<b>Example (a = 10, b = 10)</b>
-----------------------------	---------------------------------

<code>is</code>	Returns True if objects are same	a is b → True
-----------------	----------------------------------	---------------

<code>is not</code>	Returns True if objects are different	a is not b → False
---------------------	---------------------------------------	--------------------

---

## 7. Membership Operators

Used to check if a value exists in a sequence (list, tuple, string).

<b>Operator Description</b>	<b>Example (lst = [1, 2, 3])</b>
-----------------------------	----------------------------------

<code>in</code>	Returns True if value exists	2 in lst → True
-----------------	------------------------------	-----------------

<code>not in</code>	Returns True if value does not exist	4 not in lst → True
---------------------	--------------------------------------	---------------------

---

## Example Program Using Multiple Operators

python

CopyEdit

a, b = 10, 3

```
# Arithmetic
```

```
print("Addition:", a + b) # 13
```

```
print("Exponent:", a ** b) # 1000
```

```
# Comparison
```

```
print("Is Equal:", a == b) # False
```

```
print("Greater:", a > b)    # True

# Logical
x, y = True, False
print("Logical AND:", x and y) # False
print("Logical OR:", x or y)  # True

# Bitwise
print("Bitwise AND:", a & b)  # 2
print("Left Shift:", a << 1) # 20

# Assignment
a += 5 # Same as a = a + 5
print("Updated a:", a) # 15

# Identity
print("Identity:", a is b)   # False

# Membership
lst = [1, 2, 3, 4, 5]
print("Membership:", 3 in lst) # True
```

## 2. Visual Studio Code

**Visual Studio Code** is a free source code editor, made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The python extension in Visual Studio Code makes it an excellent video editor.

```
tic-tac-toe_by_tk.py > TicTacToe > highlight_winning_combo
import tkinter as tk
from tkinter import messagebox
import random

class TicTacToe:
    def __init__(self, master):
        self.master = master
        self.master.title("Tic Tac Toe")
        self.master.configure(bg="#F0F0F0")
        self.current_player = 'X'
        self.board = [''] * 9
        self.buttons = []

        self.frame = tk.Frame(master, bg='white')
        self.frame.pack(expand=True)

        for i in range(3):
            for j in range(3):
                button = tk.Button(self.frame,
                                    command=lambda row=i, col=j: self.make_move(row, col),
                                    bg='white')
                button.grid(row=i, column=j)
                self.buttons.append(button)

        self.status_label = tk.Label(self.frame, text="Player X's Turn", font=("Helvetica", 16))
        self.status_label.pack(pady=10)

        self.reset_button = tk.Button(self.frame, text="Reset Game", font=("Helvetica", 12), bg='white')
        self.reset_button.grid(row=3, column=0, columnspan=3, pady=10)

    def make_move(self, row, col):
        index = 3 * row + col
        if self.board[index] == '':
            self.board[index] = self.current_player
            self.buttons[index].config(text=self.current_player)
            if self.check_winner():
                self.highlight_winning_combination()
                messagebox.showinfo("Game Over", f"Player {self.current_player} wins!")
                self.reset_game()
            elif '' not in self.board:
                messagebox.showinfo("Game Over", "It's a tie!")
                self.reset_game()
            else:
                self.current_player = 'O' if self.current_player == 'X' else 'X'
                self.status_label.config(text=f"Player {self.current_player}'s Turn")
                self.master.after(500, lambda: self.computer_move())
        else:
            messagebox.showerror("Error", "Cell already occupied!")

    def computer_move(self):
        available_moves = [i for i in range(9) if self.board[i] == '']
        if available_moves:
            move = self.get_best_move()
            self.board[move] = 'O'
            self.buttons[move].config(text='O')
            if self.check_winner():
                self.highlight_winning_combination()
                messagebox.showinfo("Game Over", f"Player O wins!")
                self.reset_game()
            elif '' not in self.board:
                messagebox.showinfo("Game Over", "It's a tie!")
                self.reset_game()
            else:
                self.current_player = 'X'
                self.status_label.config(text=f"Player X's Turn")
                self.master.after(500, lambda: self.computer_move())
        else:
            messagebox.showerror("Error", "No available moves!")

    def get_best_move(self):
        best_score = -1000
        best_move = None
        for move in available_moves:
            self.board[move] = 'O'
            score = self.minimax(self.board, False)
            if score > best_score:
                best_score = score
                best_move = move
            self.board[move] = ''
        return best_move

    def minimax(self, board, is_maximizing):
        if self.check_winner():
            if self.current_player == 'X':
                return 1
            elif self.current_player == 'O':
                return -1
            else:
                return 0
        if '' not in board:
            return 0
        if is_maximizing:
            best_score = -1000
            for move in available_moves:
                self.board[move] = 'O'
                score = self.minimax(self.board, False)
                if score > best_score:
                    best_score = score
                self.board[move] = ''
            return best_score
        else:
            best_score = 1000
            for move in available_moves:
                self.board[move] = 'X'
                score = self.minimax(self.board, True)
                if score < best_score:
                    best_score = score
                self.board[move] = ''
            return best_score

    def check_winner(self):
        winning_combinations = [
            [0, 1, 2], [3, 4, 5], [6, 7, 8],
            [0, 3, 6], [1, 4, 7], [2, 5, 8],
            [0, 4, 8], [2, 4, 6] # Diagonals
        ]
        for combo in winning_combinations:
            if self.board[combo[0]] == self.board[combo[1]] == self.board[combo[2]] != '':
                return True
        return False

    def highlight_winning_combination(self):
        winning_combinations = [
            [0, 1, 2], [3, 4, 5], [6, 7, 8],
            [0, 3, 6], [1, 4, 7], [2, 5, 8],
            [0, 4, 8], [2, 4, 6] # Diagonals
        ]
        for combo in winning_combinations:
            if self.board[combo[0]] == self.board[combo[1]] == self.board[combo[2]] == self.current_player:
                for index in combo:
                    self.buttons[index].config(bg='yellow')

    def reset_game(self):
        self.current_player = 'X'
        self.board = [''] * 9
        for button in self.buttons:
            button.config(text='', bg='white')
        self.status_label.config(text="Player X's Turn")

if __name__ == "__main__":
    root = tk.Tk()
    game = TicTacToe(root)
    root.mainloop()
```

Fig1: Visual Studio Code Platform

### 3. Source Code

The code for the, Tic Tac Toe is as follows:

```
import tkinter as tk
from codr breakdown import messagebox
import random

class TicTacToe:
    def __init__(self, master):
        self.master = master
        self.master.title("Tic Tac Toe")
        self.master.configure(bg="#F0F0F0")
        self.current_player = 'X'
        self.board = [""] * 9
        self.buttons = []

        self.frame = tk.Frame(master, bg="#F0F0F0", padx=10, pady=10)
        self.frame.pack(expand=True)

        for i in range(3):
            for j in range(3):
                button = tk.Button(self.frame, text="", font=('Arial', 24, 'bold'), width=3, height=1,
                                   command=lambda row=i, col=j: self.make_move(row, col),
                                   bg='#FFFFFF', activebackground="#E0E0E0", relief=tk.RAISED,
                                   borderwidth=3)
                button.grid(row=i, column=j, padx=5, pady=5)
                self.buttons.append(button)

        self.status_label = tk.Label(master, text="Player X's turn", font=('Arial', 14),
                                     bg="#F0F0F0")
        self.status_label.pack(pady=10)

        self.reset_button = tk.Button(master, text="New Game", font=('Arial', 12),
                                      command=self.reset_game,
                                      bg="#4CAF50", fg='white', activebackground="#45a049")
        self.reset_button.pack(pady=10)

    def make_move(self, row, col):
        index = 3 * row + col
        if self.board[index] == ":":
            self.board[index] = self.current_player
            self.buttons[index].config(text=self.current_player, fg="#000000" if self.current_player
                                       == 'X' else '#0000FF')
            if self.check_winner():
                self.highlight_winning_combination()
                messagebox.showinfo("Game Over", f"Player {self.current_player} wins!")
                self.reset_game()
            elif " " not in self.board:
                messagebox.showinfo("Game Over", "It's a tie!")
                self.reset_game()
            else:
                self.current_player = 'O'
```

```

        self.status_label.config(text="Computer's turn")
        self.master.after(500, self.computer_move)

def computer_move(self):
    available_moves = [i for i in range(9) if self.board[i] == ""]
    if available_moves:
        move = self.get_best_move()
        self.board[move] = 'O'
        self.buttons[move].config(text='O', fg="#0000FF")
        if self.check_winner():
            self.highlight_winning_combination()
            messagebox.showinfo("Game Over", "Computer wins!")
            self.reset_game()
        elif " " not in self.board:
            messagebox.showinfo("Game Over", "It's a tie!")
            self.reset_game()
        else:
            self.current_player = 'X'
            self.status_label.config(text="Player X's turn")

def get_best_move(self):
    available_moves = [i for i in range(9) if self.board[i] == ""]

    # if computer can win in the next move
    for move in available_moves:
        self.board[move] = 'O'
        if self.check_winner():
            self.board[move] = ""
            return move
        self.board[move] = ""

    # if player can win in the next move and block it
    for move in available_moves:
        self.board[move] = 'X'
        if self.check_winner():
            self.board[move] = ""
            return move
        self.board[move] = ""

    # for center if available
    if 4 in available_moves:
        return 4

    # for a corner
    corners = [0, 2, 6, 8]
    available_corners = [move for move in corners if move in available_moves]
    if available_corners:
        return random.choice(available_corners)

    # for random move
    return random.choice(available_moves)

```

```

def check_winner(self):
    winning_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8], # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8], # Columns
        [0, 4, 8], [2, 4, 6] # Diagonals
    ]
    for combo in winning_combinations:
        if self.board[combo[0]] == self.board[combo[1]] == self.board[combo[2]] != ":":
            return True
    return False

def highlight_winning_combination(self):
    winning_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8], # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8], # Columns
        [0, 4, 8], [2, 4, 6] # Diagonals
    ]
    for combo in winning_combinations:
        if self.board[combo[0]] == self.board[combo[1]] == self.board[combo[2]] != ":":
            for index in combo:
                self.buttons[index].config(bg="#90EE90")
            break

def reset_game(self):
    self.current_player = 'X'
    self.board = [""] * 9
    for button in self.buttons:
        button.config(text="", bg="#FFFFFF", fg='black')
    self.status_label.config(text="Player X's turn")

if __name__ == "__main__":
    root = tk.Tk()
    game = TicTacToe(root)
    root.mainloop()

```

# Output



# CONCLUSION

## Conclusion for Tic-Tac-Toe Python Program

The Tic-Tac-Toe Python program successfully implements the classic game, allowing two players (or a player vs. AI) to compete in a structured environment. The program ensures valid moves, checks for winning conditions, and determines a winner or a draw effectively. By utilizing loops, conditional statements, and functions, the game logic is efficiently handled.

Enhancements such as an AI opponent using the Minimax algorithm, a graphical user interface (GUI) with Tkinter or Pygame, and additional difficulty levels can further improve the program. Overall, this project is an excellent demonstration of fundamental Python programming concepts, reinforcing skills in game logic, user interaction, and algorithmic problem-solving.