BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Department of Computer Science & Engineering
CSE-100
Project Report

A Project called Tic-Tac-Toe game using C Programming

Submitted To
Iffat Tamanna
Assistant Professor
Dept. Of CSE

Submitted By
Mst.Sumaiya Akter (ID: 22235103059)
Utsho Roy (ID: 22235103261)
Md.Shamim Pramanik (ID: 22235103267)
Afrin Akter Achol (ID: 22235103358)

Date : 06-12-2023

Contents

Chapter 1

Introduction

1.1 Introduction
In this section, the user is prompted to choose between the two game modes. The code then initializes the game based on the selected mode, with additional instructions and information provided for clarity.

1.2 Project Scope
The project scope is well-defined, covering the implementation of a Tic-Tac-Toe game with two distinct modes. The scope includes features such as player turns, marking the board, checking for a winner, and handling player vs computer interactions.

1.3 Objective
The main objectives of the code include creating a functional Tic-Tac-Toe game, allowing players to compete either against each other or against the computer. The objectives are met through functions like **markingBoard(), checkWinner(), comp_win(), and block_user()**.

1.4 Conclusion
The conclusion section of the introduction summarizes the user's choice and acknowledges the completion of the game. It mentions the winner (if any) or declares a draw. Additionally, the code implements color-coded output for 'X' and 'O' to enhance the user interface.

The provided code successfully achieves its objectives, providing an interactive Tic-Tac-Toe gaming experience. However, it's worth noting that the code could benefit from some improvements, such as organizing repetitive code into functions to enhance readability and maintainability. Additionally, comments within the code could be added to explain the logic and enhance code documentation.

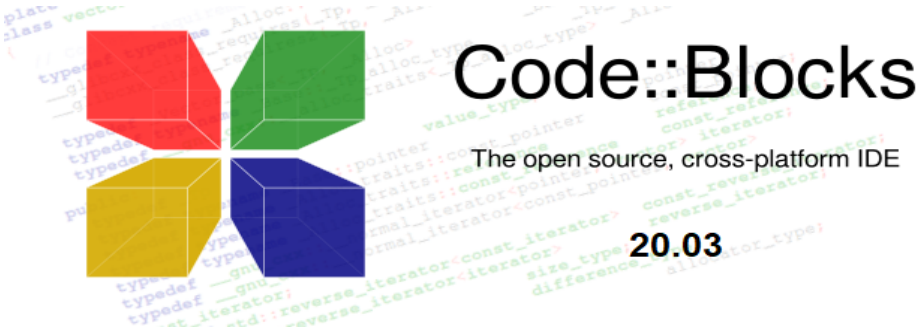
Chapter 2
Analysis

2.1 Existing System Analysis
The existing system is a Tic-Tac-Toe game implemented in C, allowing players to play against each other or against the computer.
The code uses a simple console-based interface for interaction.
The game provides basic visual enhancements with colored output for 'X' and 'O'.

2.2 Feasibility Study
The feasibility study typically involves assessing whether the project is viable in terms of technical, operational, and economic aspects.
From a technical standpoint, the Tic-Tac-Toe game is feasible, given the simplicity of the game and the available technology (C programming language).
Operationally, the game functions, but there are logical issues that need to be addressed for better gameplay and user experience.
Economically, since this is a small-scale project, the feasibility is reasonable.

Chapter 3
Used Software and Programming Language

3.1 Code::Blocks
Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++.
It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C, C++, and Fortran. It has a custom build system and optional Make support.



After releasing two release candidate versions, 1.0rc1 on July 25, 2005 and 1.0rc2 on October 25, 2005, instead of making a final release, the project developers started adding many new

features, with the final release being repeatedly postponed. Instead, there were nightly builds of the latest SVN version made available on a daily basis. [citation needed] The first stable release was on February 28, 2008, with the version number changed to 8.02. The versioning scheme was changed to that of Ubuntu, with the major and minor number representing the year and month of the release. Version 17.12 is the latest stable release; however, for the most up-to-date version the user can download the relatively stable nightly build or download the source code from SVN. Jennic Limited distributes a version of Code::Blocks customized to work with its microcontrollers.[Wikipedia]

3.2 C Programming Language

What is C?

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language, despite being old. The main reason for its popularity is because it is a fundamental language in the field of computer science.

C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

Why Learn C?

- It is one of the most popular programming language in the world
- If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar
- C is very fast, compared to other programming languages, like Java and Python
- C is very versatile; it can be used in both applications and technologies

Difference between C and C++

- C++ was developed as an extension of C, and both languages have almost the same syntax
- The main difference between C and C++ is that C++ support classes and objects, while C does not

Chapter 4
Code Structure and Purpose

4.1 Header Inclusion
The code includes the standard input-output header #include <stdio.h>.

4.2 Global Variables
The global variable box is declared as an array of characters, representing the tic-tac-toe board. Another global variable mark is declared but not initialized.

4.3 Function Declarations
4.3.1 createBoard(): Displays the tic-tac-toe board.

Here's a breakdown of how the function works:

1. **Printing the Header:**
   - The function begins by printing the title "Tic-tac-toe" and information about the players (Player 1 with 'X' and Player 2 with 'O').
2. **Printing the Board Layout:**
   - The Tic-Tac-Toe board is represented as a grid with 3 rows and 3 columns.
   - The board positions are represented by elements in the box array, which contains characters '1' to '9'.
   - The **printColored()** function is used to print the characters in color. It prints 'X' in green and 'O' in red, while other characters are printed normally.
   - The board is displayed with horizontal and vertical lines to separate the cells.
3. **Printing Rows:**
   - Each row of the board is printed separately, with lines separating the rows.
4. **Printing Colored Marks:**
   - The **printColored()** function is called for each position on the board to print the corresponding mark with color.

Here's an overview of the code structure inside the function:

```
void createBoard()
{
   printf("\n\n\t\tTic-tac-toe\n\n");
   printf("\t\t\t\t\tPlayer 1 (X) -- Player 2 (O)\n\n");
   printf("\t\t\t\t\t\t     |    |    \n");
   // ... Printing rows with colored marks
   printf("\t\t\t\t\t\t_____|____|____\n");
   // ... Printing remaining rows
   printf("\t\t\t\t\t\t     |    |    \n");
   // ... Printing the last row
}
```

This function is called within the main game loop to display the updated game board after each move by a player.

4.3.2 markingBoard(int choice, char mark): Marks the board based on the user's or cmputer's choice.

Here's an explanation of how the function works:

1. **Input Validation:**
   - The function first checks if the chosen position (choice) is a valid position on the board and if the corresponding element in the box array is equal to its position number ('1' to '9'). If the chosen position is valid, the function proceeds to mark the board; otherwise, it prints "Invalid."
2. **Marking the Board:**
   - If the chosen position is valid, the function updates the game board by assigning the mark character to the corresponding position in the box array.
   - The **printColored()** function is then called to print the updated mark on the board in color.

Here's an overview of the code structure inside the function:

```
void markingBoard(int choice, char mark)
{
   if (choice == 1 && box[1] == '1')
   {
      box[1] = mark;
      printColored(mark);
   }
   else if (choice == 2 && box[2] == '2')
   {
      box[2] = mark;
      printColored(mark);
   }
   .
   .
   .
   .
   .
   else if (choice == 8 && box[8] == '8')
   {
      box[8] = mark;
      printColored(mark);
   }
   else if (choice == 9 && box[9] == '9')
   {
      box[9] = mark;
      printColored(mark);
   }
```

```
    else
        printf("Invalid");
}
```

This function is typically called within the main game loop, allowing players to make moves and updating the game board accordingly. The **printColored()** function is used to display the marks in different colors based on the player ('X' in green and 'O' in red).

4.3.3 checkWinner(): Checks for a winner or a draw.

Here's an explanation of how the checkWinner() function works:

1. **Horizontal Check:**
   - It first checks for a win in each horizontal row by comparing the values in consecutive positions (1-2-3, 4-5-6, 7-8-9).
2. **Vertical Check:**
   - Next, it checks for a win in each vertical column by comparing the values in positions (1-4-7, 2-5-8, 3-6-9).
3. **Diagonal Check:**
   - It checks for a win in both diagonal directions: from the top-left to the bottom-right (1-5-9) and from the top-right to the bottom-left (3-5-7).
4. **Draw Check:**
   - If none of the win conditions are met, the function checks if all positions on the board are filled. If they are, the game is considered a draw.
5. **Return Values:**
   - If any player has won, the function returns 1.
   - If the game is a draw, meaning all positions are filled and there is no winner, the function returns 0.
   - If the game is still ongoing, the function returns -1.

Here's the code structure of the checkWinner() function:

```
int checkWinner()
{
    if (box[1] == box[2] && box[2] == box[3])  // Horizontal check
        return 1;
    else if (box[4] == box[5] && box[5] == box[6])  // Horizontal check
        return 1;
    else if (box[7] == box[8] && box[8] == box[9])  // Horizontal check
        return 1;
    else if (box[1] == box[4] && box[4] == box[7])  // Vertical check
        return 1;
    // ... Continue vertical and diagonal checks
    else if (box[1] == box[5] && box[5] == box[9])  // Diagonal check
        return 1;
    else if (box[3] == box[5] && box[5] == box[7])  // Diagonal check
        return 1;
```

```
    else if (box[1] != '1' && box[2] != '2' && /*...*/ box[9] != '9')  // Draw check
        return 0;

    else
        return -1;  // Game still ongoing
}
```

This function is typically called within the main game loop after each move to check if there's a winner or if the game has ended in a draw. The return value guides the flow of the game to either continue, declare a winner, or declare a draw.

4.3.4 comp_win(): Checks if the computer can win in the next move.

Here's an explanation of how the comp_win() function works:

1. **Winning Move Check:**
   - The function checks for various winning patterns for the computer, which is represented by the mark 'O'.
   - It looks for two 'O' marks in a row and an empty position in the third spot, allowing the computer to complete a winning line.
2. **Move Execution:**
   - If a winning move is found, the function updates the game board by placing an 'O' in the position that completes the winning line.
   - If no winning move is possible, the function returns 0, indicating that the computer did not make a winning move on this turn.

Here's the code structure of the comp_win() function:

```
int comp_win()
{
    // Check for winning moves and update the board accordingly
    if (box[1] == 'O' && box[2] == 'O' && box[3] == '3')
    {
        box[3] = mark;
        return 1;
    }
    else if (box[1] == 'O' && box[3] == 'O' && box[2] == '2')
    {
        box[2] = mark;
        return 1;
    }
    // ... Continue checking other winning move patterns

    // If no winning move is found
    return 0;
}
```

Please note that the **comp_win()** function seems to have a limited set of winning move patterns hardcoded in the function. It checks specific patterns and makes

moves accordingly. If you want to enhance the computer's strategy, you might need to consider a more dynamic approach or use artificial intelligence techniques to make the computer play more intelligently.

4.3.5 block_user(): Checks if the computer can block the user from winning.

Here's an explanation of how the block_user() function works:

1. **Blocking Move Check:**

   - The function checks for various patterns where the human player (represented by the mark 'X') has two marks in a row and an empty position in the third spot. This is a situation where the human player is one move away from winning.

2. **Move Execution:**

   - If a potential winning move by the human player is found, the function updates the game board by placing an 'O' (computer's mark) in the position that blocks the potential win.
   - If no potential winning move by the human player is detected, the function returns 0, indicating that the computer did not make a blocking move on this turn.

Here's the code structure of the block_user() function:

```
int block_user()
{
   // Check for potential winning moves by the user and block them
   if (box[1] == 'X' && box[2] == 'X' && box[3] == '3')
   {
      box[3] = mark;
      return 1;
   }
   else if (box[1] == 'X' && box[3] == 'X' && box[2] == '2')
   {
      box[2] = mark;
      return 1;
   }
   // ... Continue checking other potential winning move patterns

   // If no potential winning move by the user is found
   return 0;
}
```

Similar to the **comp_win()** function, the **block_user()** function has a hardcoded set of patterns that it checks to determine if the human player is about to win. It then makes a move to block that potential win. If you want to enhance the computer's strategy, you might consider a more dynamic approach or use AI techniques for a more intelligent gameplay strategy.

4.3.6 printColored(char c): The color green is used for 'X', and the color red is used for 'O'.

Here's an explanation of how the printColored function works:

1. **Coloring the Output:**
   - The function takes a character c as its parameter, representing the mark to be printed on the console.
   - It uses ANSI escape codes to change the text color based on the value of c.
2. **Color Codes:**
   - If 'c' is equal to 'X', the function uses the escape code \033[32m to set the color to green. This represents an 'X' mark.
   - If 'c' is equal to 'O', the function uses the escape code \033[31m to set the color to red. This represents an 'O' mark.
   - If 'c' is any other character, it prints the character normally without changing the color.
3. **Resetting Color:**
   - After printing the colored character, the function uses the escape code \033[0m to reset the color to the default.

Here's the code structure of the printColored function:

```
void printColored(char c)
{
   if (c == 'X')
      printf("\033[32m%c\033[0m", c); // Green color for 'X'
   else if (c == 'O')
      printf("\033[31m%c\033[0m", c); // Red color for 'O'
   else
      printf("%c", c);
}
```

This function is called within the **createBoard()** and **markingBoard()** functions to print the Tic-Tac-Toe board with colored 'X' and 'O' marks. The use of colors enhances the visual representation of the game on the console.

Chapter 5

Main Function

The main function provides a menu to choose between Player vs Computer and Player vs Player modes. Depending on the mode selected, the appropriate game logic is executed.

5.1 Game Modes

1. ***Player vs Computer:***
   - The player and computer take turns making moves.

- The computer has a simple strategy to either win or block the player from winning.

2. *Player vs Player:*

- Two players take turns making moves until a winner or a draw is determined.

5.2 Code Analysis

Player Input Handling

- The code handles player input and validates it for correctness.
- Invalid moves prompt the player to enter a valid move.

Tic-Tac-Toe Board Display

- The createBoard() function is responsible for displaying the current state of the board.

Winning Conditions

- Winning conditions are checked for rows, columns, and diagonals.
- The game continues until a winner is determined or there is a draw.

Computer AI

- The computer has a basic strategy to either win or block the player.

Code Organization

- The code is organized with clear functions and comments, making it relatively easy to follow.

5.3 Suggestions for Improvement

Randomization for Computer Moves

- Introduce some level of randomness in the computer's moves to make the game more dynamic.

Function Decomposition

- Break down larger functions, such as comp_win() and block_user(), into smaller, more modular functions to improve readability and maintainability.

User Interface

- Enhance the user interface with clear instructions and better formatting for a more user-friendly experience.

Code Optimization:

- Some parts of the code could be optimized for better efficiency, but this depends on specific requirements.

Error Handling:

- Include more detailed error messages for invalid inputs to guide the user on what went wrong.

Chapter 6
Output

6.1 Player vs Computer

Choose Game Mode:
1. Player vs Computer
2. Player vs Player

Enter your choice : 1

You selected Player vs Computer
Here player 2 is the Computer

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

```
     |   |
  1  | 2 | 3
_____|___|_____
     |   |
  4  | 5 | 6
_____|___|_____
     |   |
  7  | 8 | 9
_____|___|_____
     |   |
```

Player, enter a number: 5

Tic-tac-toe

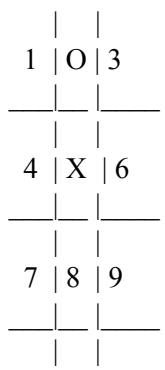
Player 1 (X) -- Player 2 (O)

```
     |   |
  1  | 2 | 3
_____|___|_____
     |   |
  4  | X | 6
_____|___|_____
     |   |
  7  | 8 | 9
_____|___|_____
     |   |
```

Computer, enter a number

Tic-tac-toe

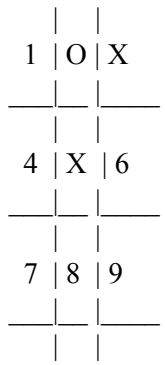
Player 1 (X) -- Player 2 (O)



Player, enter a number: 3

Tic-tac-toe

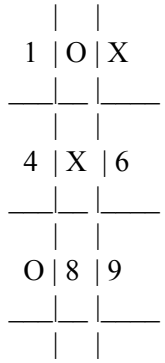
Player 1 (X) -- Player 2 (O)



Computer, enter a number

Tic-tac-toe

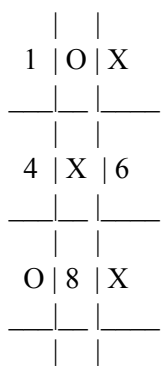
Player 1 (X) -- Player 2 (O)



Player, enter a number: 9

Tic-tac-toe

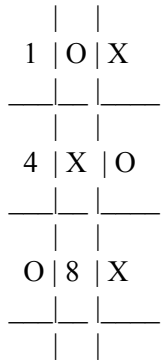
Player 1 (X) -- Player 2 (O)



Computer, enter a number

Tic-tac-toe

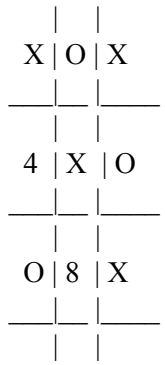
Player 1 (X) -- Player 2 (O)



Player, enter a number: 1

Tic-tac-toe

Player 1 (X) -- Player 2 (O)



Player 1 You have won the game

6.2 Player vs Player

Choose Game Mode:
1. Player vs Computer
2. Player vs Player

Enter your choice : 2

You selected Player vs Player

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

```
     |     |
  1  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  9
_____|_____|_____
     |     |
```

Player 1, enter a number:1

X

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

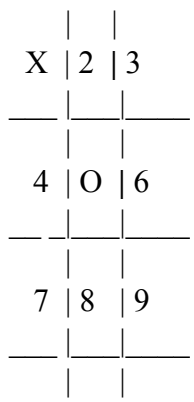
```
     |     |
  X  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  9
_____|_____|_____
     |     |
```

Player 2, enter a number:5

O

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

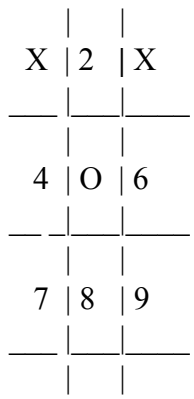


Player 1, enter a number:3

X

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

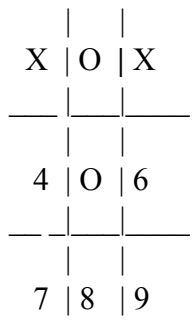


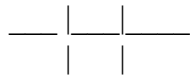
Player 2, enter a number:2

O

Tic-tac-toe

Player 1 (X) -- Player 2 (O)



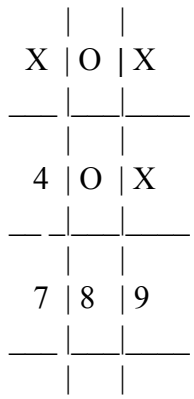


Player 1, enter a number:6

X

Tic-tac-toe

Player 1 (X) -- Player 2 (O)

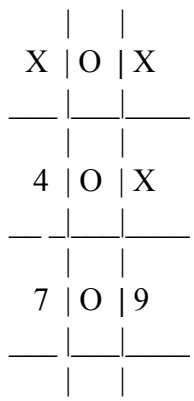


Player 2, enter a number:8

O

Tic-tac-toe

Player 1 (X) -- Player 2 (O)



Player 2 You have won the game

Chapter 7
Limitations, Future Work & Conclusion

7.1 Limitations
There is always some limitation in every project. We also have limitations in our Tic-Tac-Toe game. And those limitations are:

- It doesn't have 4x4,5x5 up to 13x13 box features.
- It doesn't have storage system that can store wining point(score) of player.
- It doesn't have high graphical system and graphical user interface (GUI).
- User can't change background color because of no background changing option.
- User can't play against online players.

7.2 Future Work
In future if we upgrade our Tic-Tac Toe game. We would like to add some features that will make it better for user. And those features that we would like to add are:

- More option for boxes like 4x4,5x5 up to 13x13.
- Player will be able to change their background music & color.
- Score storing system that can store player wining point. 41
- Online multi player system where player will be able play against online players.

7.3 Conclusion
The provided code successfully implements a basic tic-tac-toe game with two modes (Player vs Computer and Player vs Player). The game logic and player input handling are generally sound, but there's room for improvement in terms of user interface and code organization.

Chapter 8
References

- Teach yourself C by herbert schildt.
- C Programming -Absolute Beginner's Guide by by Greg Perry and Dean Miller
- https:///programming-tic-tac-toe-in-c-6ba4b6965ba3
- https:///software-engineering-agile-development-models/
- https:///blog/how-to-create-a-project-proposal
- https:///wiki/Tic-tac-toe