BRADLEY
UNIVERSITY

ECE 103: Intro to EE: Computers and Programming

Tic-Tac-Toe Game

Grant Abella, Alex Jaeger, Ethan Horvath

Contact Information

Grant Abella
Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University

Williams Hall 340
821 N. Duryea Place
Peoria, IL, 61606, USA

Phone: +1 (630) 776-4026
e-Mail: gabella@mail.bradley.edu

Table of Contents

1 Project proposal

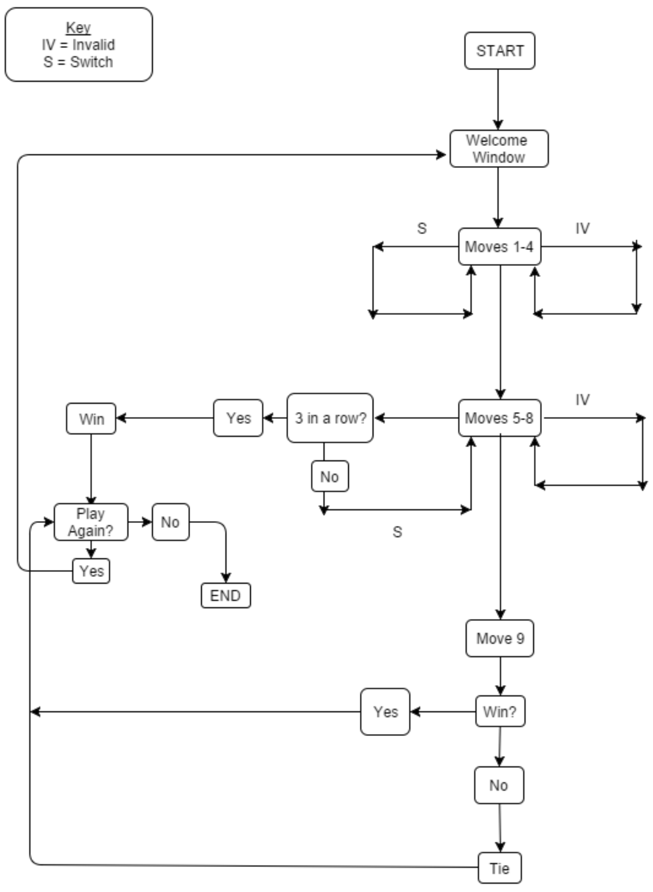
1.1 Overview

We have chosen to program a Tic-Tac-Toe game because we believe that it will provide a nice challenge and completing it will be rewarding. Now that we have completed the C programming portion of ECE 103, we know that we have the skills to program this game successfully.

1.2 Specifications

- The user will a computer that is running a version of windows that is compatible with Visual Studio.
- The user will also need a keyboard to be able to input information into the console window.
- A monitor will also be necessary so that the user can see the game running.

1.3 Block Diagram



.

At the beginning of the program, player 1 will have the first move with the character "x." The program asks the player to enter in x and y coordinates, respect to the game board. If the user enters in coordinates that are invalid (0 < x or y < 4) or enter a coordinate that has already been entered into the game from a previous turn, it will be an invalid move and reset the entry, allowing the user to try again.

If the move is valid, it will go to the next move and switch players. Once the program has reached move 5, player 1 has the opportunity to win the game. The same process for recognizing invalid moves is still in use, but now it will look for three of the same characters in a row in the horizontal, vertical, and diagonal directions.

If there are not three in a row, it will go to the next move and switch players. If there are three in a row, however, it will end the game and ask if the user would like to play again. If they select yes, it will restart at the beginning, but if no is selected, the program will shut down.

If the game continues to move 9, the game will be forced to go the "Play again?" option due to either a win or tie.

1.4 Division of Labor

Grant Abella will program the game to handle and interpret input from the user. He will also program the game's output to the console window.

Alex Jaeger will program the game to check for a win after the user inputs their choice. He will also handle some of
the other miscellaneous game logic.

Ethan Horvath will program the functions that handle manipulating and interpreting the game board. He is also responsible for making sure that the overall user-experience is positive.

1.5 Timeline and milestones

By April 15th we will have the game display working and handling input from the user.

By April 20th we will have the game programmed to look for three X's or three O's in a row.

By April 25th we will have the entire game finished and will look for different ways to improve the game since we will have a week to make adjustments to it before we have to demonstrate the game.

2 Project Report

2.1 Introduction

- Motivation:
  Our motivation for this project was that we wanted to create a Tic-Tac-Toe game that was easy to use and fun for people to play together. We knew that part of the challenge in programming games is making them user-friendly enough that players can easily tell what they need to do in order to win the game. Another big motivation for this project was being able to program the game in a concise and efficient way, but still maintaining the quality of a well-written program. Fortunately, our C code for this game came in at just over 420 lines and our Python code was just under 220 lines. Even with this relatively small amount of code for both versions, the game is bug free and very user-friendly. We are pleased to say that we feel we have successfully accomplished everything that we initially set out to do concerning this game, and even added in some extra things on top of that.

- Problem description:
  The problem that we were trying to solve was that the game needed to be programmed in a way that was easy for the player to use. To solve this, we maintained a basic layout for both the C and Python versions of the game. We believe this allows users of all types to be able to easily play the game even if they have very limited computer knowledge. We also had to make sure that our game remained faithful to the classic game of Tic-Tac-Toe that we are all used to. This was accomplished by doing our best to keep the feel of playing with pen and paper through the console window interface. The game layout is simple, easy to understand, and keeps the original look of the classic game.

- Report outline:
  The following report outlines the ins and outs of the Tic-Tac-Toe game, including background information about the project, our design procedure that we followed, the project results, and a final summary.

2.2 Background

The resources that helped us with our project were topics we learned from Introduction to C Programming instructed by Dr. Suruz Miah, and the fourth edition of "Programming in C" by Stephen G. Kochan. The two key topics that we needed for this project were the concepts of nested for loops and arrays. As far as creating the game properly, we were able to find directions of how to play the game very easily and create a program based on those rules.

2.3 Hardware

The majority of this project solely involved software. Because we were mainly concerned with programming the game to output to the Windows console, we were not worried about having to adapt the program to any kind of standalone hardware like a BeagleBone board. Since both the C and Python versions of the game run in the system console window, the user doesn't need a high-powered machine to run it. The only hardware that is necessary to play this game is a computer running a recent version of Windows, a monitor, and a keyboard.

2.4 Design Procedure

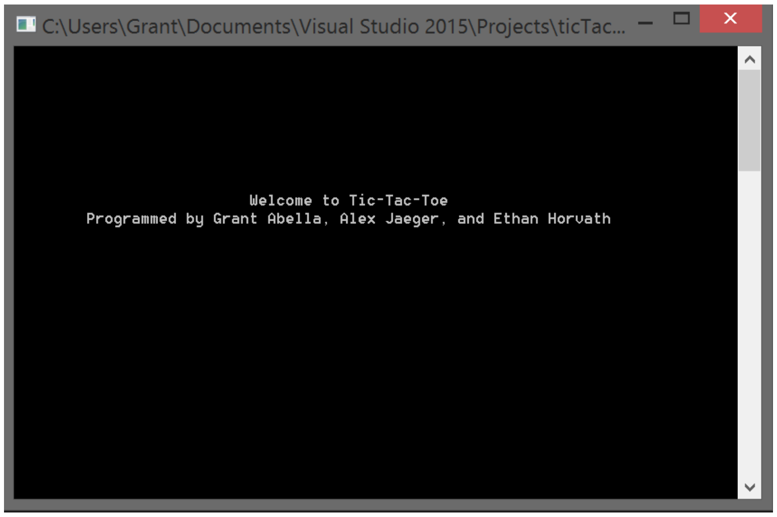
Our general procedure was as follows:

1. Plan out the user-interface
   a. Before we even began coding, a good amount of time was spent planning out how the game interface would look. This is when we decided that we wanted to use X and Y axes to notated the open spaces on the game board. These coordinates are how the user tells the program which space they would like to mark when it is their turn. We decided to use this method because it was the easiest way that we could let the user tell the program where he or she would like to make a mark on the game board.
   b. In addition to the game board and coordinates, we also decided to add in some extra things such as a win count for both players, and a small bit of text to notate whose turn it currently is. We added these things because they were not hard to program and they added a little bit more to the game. The win counters allow players to become more competitive with the game by showing who has more wins that the other. We felt that this adds a little more replay-value to the game.
2. Handle input and output
   a. We knew that we needed some way to prompt the user for input, and then a way to take that input into the program. For this we used simple print statements and functions such as scanf and input in C and Python respectively.
   b. Since the program takes place entirely in the system console window, we did not have much difficulty displaying the game board and various messages to the user. The only issue in this area was with Python's print statements. By default, when Python prints to the screen, it automatically jumps to the next line in the console window. This proved to be a problem for us because our method for printing the game board was to print each character individually and go to the next line when the end of a row in the character list had been reached. During our first test run of our Python code, we were surprised to see that the board had been printed to the console, but was stretched in a single-character column along the screen. To fix this, we had to implement a print function found in the sys library. This function does not jump to a new line after printing, so it served our purposes well.
3. Program a win-detection system
   a. This portion of the design process was the most difficult as it involved programming the game to check for three X's or O's in a row, column, or diagonally across the board. To accomplish this, we used a method similar to how we output the game board to the console. We had the program traverse the board space-by-space and test the contents of each board space. If a player's mark had been detected in a space, the program adds one to a mark counter. By the end of a row, column, or diagonal, the counter is tested to see if it is equal to three. If it is, that player wins the game. Through careful design of nested for loops, we successfully employed this system to check for a win in all possible paths on the board.

b. The second part of the win-detection system was asking the user if they wished to play again. If so, we needed to program a way to reset the game so that he or she could play the game again. This was easily accomplished by writing functions that fill the game board with empty spaces and increment the win counter for a player if he or she won the previous game. By accomplishing this objective, we added an element of replay value to the game because it could be played multiple times in one sitting.

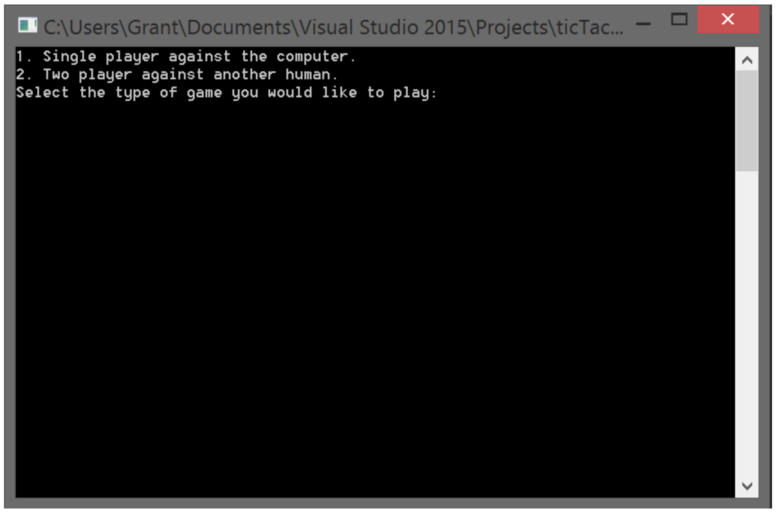
4. Smooth out the user experience and fix bugs
   a. Programming the game did not come without any challenges. There were numerous times when the game results were not what we desired and we were forced to go back and debug the game until it was perfect. As described in the input and output section above, we had to solve the issue of printing the game board to the screen with Python. Another challenge that arose was that the indices for the game board array did not correlate to the labeled coordinates in the game. This was because the lines denoting the borders of the game board also took up a space in the character array. To solve this problem, we defined functions that adjusted the values that the user inputs so that they accurately reflect the indices of the array.
5. Adapt our C code to Python
   a. The transition from C to Python was fairly smooth. Since the only real difference is the syntax we were able to translate the basic layout and program flow of our C code into Python. The biggest differences were Python's methods for getting input and output and that the game board had to be a list because Python does not use arrays. Other than that, the transition from C to Python was straightforward and relatively smooth.

2.5 Implementation Results



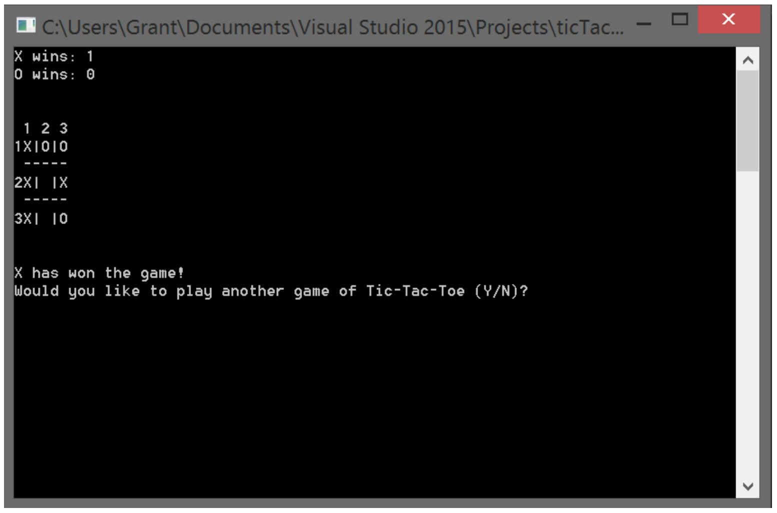
```
C:\Users\Grant\Documents\Visual Studio 2015\Projects\ticTac...

                          Welcome to Tic-Tac-Toe
          Programmed by Grant Abella, Alex Jaeger, and Ethan Horvath
```

The Tic-Tac-Toe welcome screen with credits to the three programmers. The game progresses to the next screen when any key is pressed.



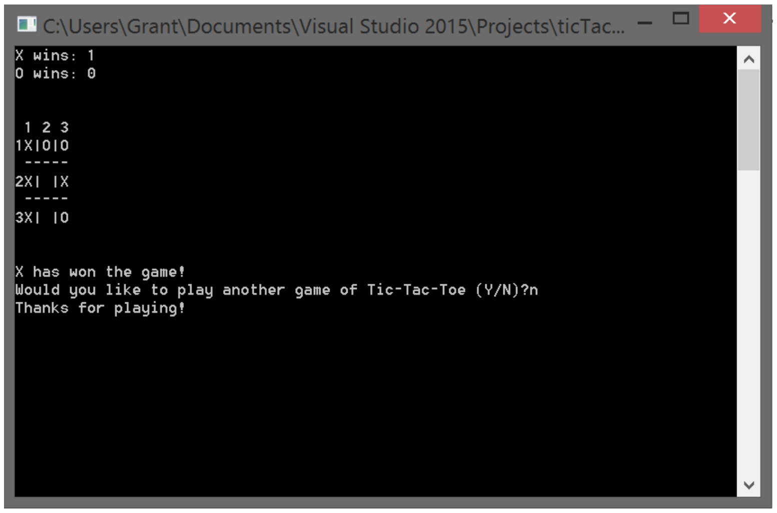
```
C:\Users\Grant\Documents\Visual Studio 2015\Projects\ticTac...
1. Single player against the computer.
2. Two player against another human.
Select the type of game you would like to play:
```

The second screen of the game prompts the user to choose a game mode. In single player mode, the user plays against the computer. In two player mode, the user plays against another player.



```
X wins: 1
O wins: 0


 1 2 3
1X|O|O
 -----
2X| |X
 -----
3X| |O


X has won the game!
Would you like to play another game of Tic-Tac-Toe (Y/N)?
```

When a player gets three X's or O's in a row, they win the game, their win counter increases, and the game prompts them to play again.



```
X wins: 1
O wins: 0


 1 2 3
1X|O|O
 -----
2X| |X
 -----
3X| |O


X has won the game!
Would you like to play another game of Tic-Tac-Toe (Y/N)?n
Thanks for playing!
```

If the user enters 'Y' or 'y', the game is reset with a clear board, if he or she enters 'N' or 'n', the game thanks them for playing and then exits the console.

2.6 Summary and Conclusion

It is clear that this project has taught us things that we will carry with us beyond our education and into the work place. We encountered many challenges and hardships when working on our project. Overcoming adversity and various obstacles taught us how to conquer difficult problems and ultimately strengthened our knowledge, experience, and skills as electrical engineers. Going forward, our ability to program and troubleshoot problems will improve as time passes on and soon all of our hard endeavors will be a piece of cake.