**INTRODUCTION**

Tic-tac-toe (or Nougats and crosses, Xs and Os) is a [paper-and-pencil game](http://en.wikipedia.org/wiki/Paper-and-pencil_game) for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

The simplicity of Tic-tac-toe makes it ideal as a [pedagogical](http://en.wikipedia.org/wiki/Pedagogical) tool for teaching the concepts of good [sportsmanship](http://en.wikipedia.org/wiki/Sportsmanship) and the branch of artificial that deals with the searching of [game trees](http://en.wikipedia.org/wiki/Game_tree). It is straight

forward to write a [computer program](http://en.wikipedia.org/wiki/Computer_program) to play Tic-tac-toe perfectly, to enumerate the 765 essentially different positions (the [state space complexity](http://en.wikipedia.org/wiki/State_space_complexity)), or the 26,830 possible games to rotations and reflections (the [game tree complexity](http://en.wikipedia.org/wiki/Game_tree_complexity)) on this space.

Board games, which could be as simple as Tic-Tac-Toe, are where human wisdom has shined since a long time ago and are what human intelligence is trying to implement for machine intelligence for decades. Many of machine learning techniques have been explored, exploited, developed and argued extensively as researchers work on these canonical problems.

Tic-tac-toe is traditionally a popular board game among kids: in its 3 by 3 board two persons alternately place one piece at a time; one wins when he or she has three pieces of his or her own in a row, whether horizontally, vertically, or diagonally. This work will employ reinforcement learning methods in its version of “after state” evaluation to implement simple board games such as Tic-Tac-Toe.

In Tic-Tac-Toe, there are two players and a three-by-three grid. Initially the grid is empty. Each player moves in turn by placing a marker in an open square. By convention, the first player’s marker is “X” and the second player’s marker is “O”. The first player moves first. The object of the game is to place three markers in a row. This results in a win for that player and a loss for the opponent. Failing a win, a draw may be earned by preventing the opponent from placing three markers in a row. It can be shown by enumeration the game tree that at least a draw can be forced by the second player.

There were nine input and output units. Each corresponded to a square in the grid. An “X” was denoted by the value 1.0, an “O” was denoted by the value -1.0, and an open space was denoted by the value 0.0. A move was determined by presenting the current board pattern to the network and examining the relative strengths of the nine output nodes.

**Tools/Technologies to be used for implementation of Project**

The requirement analysis can be further divided into two parts.

* Hardware requirements

|  |  |
| --- | --- |
| **HARDWARE PROFILE** | **HARDWARE REQUIRED** |
| Central Processing Unit | Pentium II or higher (233 MHz or higher) |
| Hard Disk Drive | 600 MB on system drive,  10 GB on disk drive |
| RAM | With minimum capacity |
| Display | VGA display |

* Software requirements

|  |  |
| --- | --- |
| **Skills required in Software** | **Software Designation** |
| Front End | C |
| Back End | MS-Acess |
| Operating System | Windows7, Window8. |

**PROJECT UTILITY**

* Decentralization view:-
  + - * What if instead of having one person playing against another, one person plays against a team of nine players, each of whom is responsible for one cell in the 3 times3 grid? In this new way of playing the game, the team has to coordinate its players, who are acting independently based on their limited information. Essentially, the solution uses a decentralized decision making, which at first seems to complicate the solution. However, surprisingly, we show that in this mode, an equivalent level of decision making ability comes from simple components that reduce system complexity
* Complete Entertainment:-
  + - * It provides full-fledged entertainment tor the people of all age and also help to improve the mind strategy.
* Tic-Tac-Toe is a simple way to give students alternative ways of exploring and expressing key ideas and using key skills.  Typically, the Tic-Tac-Toe board has 9 cells in it, like that of the game.  This can, of course, be adjusted.

**Introduction to C Language**

C is a general-purpose high level language that was originally developed by Dennis Ritchie for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972.

The Unix operating system and virtually all Unix applications are written in the C language. C has now become a widely used professional language for various reasons.

C ended up being so efficient and flexible that in 1973, Ritchie and Ken Thompson rewrote most of the UNIX operating system using C. Many previous operating systems had been written in assembly. Unlike assembly, which ties a program to a specific CPU, C’s excellent **portability** allowed UNIX to be recompiled on many different types of computers, speeding it’s adoption. C and Unix had their fortunes tied together, and C’s popularity was in part tied to the success of UNIX as an operating system.

In 1978, Brian Kernighan and Dennis Ritchie published a book called “The C Programming Language”. This book, which was commonly known as K&R (after the author’s last names), provided an informal specification for the language and became a de facto standard. When maximum portability was needed, programmers would stick to the recommendations in K&R, because most compilers at the time were implemented to K&R standards.

In 1983, the American National Standards Institute (ANSI) formed a committee to establish a formal standard for C. In 1989 (committees take forever to do anything), they finished, and released the C89 standard, more commonly known as ANSI C. In 1990 the International Organization for Standardization adopted ANSI C (with a few minor modifications). This version of C became known as C90. Compilers eventually became ANSI C/C90 compliant, and programs desiring maximum portability were coded to this standard.

In 1999, the ANSI committee released a new version of C called C99. It adopted many features which had already made their way into compilers as extensions, or had been implemented in C++.

C++ (pronounced see plus plus) was developed by Bjarne Stroustrup at Bell Labs as an extension to C, starting in 1979. C++ was ratified in 1998 by the ISO committee, and again in 2003 (called C++03, which is what this tutorial will be teaching). A new version of the standard, known as C++11 has been made available since the time these tutorials were written — updates to the tutorial to cover C++11′s additions will be made in the appendix.

The underlying design philosophy of C and C++ can be summed up as “trust the programmer” which is both wonderful, because the compiler will not stand in your way if you try to do something unorthodox that makes sense, but also dangerous, because the compiler will not stand in your way if you try to do something that could produce unexpected results. That is one of the primary reasons why knowing how NOT to code C/C++ is important — because there are quite a few pitfalls that new programmers are likely to fall into if caught unaware.

C++ adds many new features to the C language, and is perhaps best thought of as a superset of C, though this is not strictly true as C99 introduced a few features that do not exist in C++. C++’s claim to fame results primarily from the fact that it is an object-oriented language. As for what an object is and how it differs from traditional programming methods, well, we’ll cover that in just a few sections.

* Easy to learn
* Structured language
* It produces efficient programs.
* It can handle low-level activities.
* It can be compiled on a variety of computers.

**About C/C++**:

C is a general-purpose programming language initially developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs. Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. Its design provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system.

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems.

Many later languages have borrowed directly or indirectly from C, including D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, ObjectiveC, Perl, PHP, Python, Verilo and Unix's C shell. These languages have drawn many of their control structures and other basic features from C. Most of them are also very syntactically similar to C in general, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different. C++ and Objective-C started as compilers that generated C code; C++ is currently nearly a superset of C, while Objective-C is a strict superset of C.

Before there was an official standard for C, many users and implementors relied on an informal specification contained in a book by Dennis Ritchie and Brian Kernighan; that version is generally referred to as "K&R" C. In 1989 the American National Standards Institute published a standard for C (generally called "ANSI C" or "C89"). The next year, the same specification was approved by the International Organization for Standardization as an international standard (generally called "C90"). ISO later released an extension to the internationalization support of the standard in 1995, and a revised standard in 1999. The current version of the standard (now known as "C11") was approved in December 2011.

The C and C++ programming languages are closely related. C++ grew out of C, as it was designed to be source-and-link compatible with C. Due to this, development tools for the two languages (such as IDEs and compilers) are often integrated into a single product, with the programmer able to specify C or C++ as their source language. However, due to minor semantic differences, most non-trivial C programs will not compile as C++ code without modification — C++ is not asuperset of C.

Likewise, C++ introduces many features that are not available in C and in practice almost all code written in C++ is not conforming C code. This article, however, focuses on differences that cause conforming C code to be ill-formed C++ code, or to be conforming/well-formed in both languages, but to behave differently in C and C++.

Bjarne Stroustrup, the creator of C++, has suggestedthat the incompatibilities between C and C++ should be reduced as much as possible in order to maximize inter-operability between the two languages. Others have argued that since C and C++ are two different languages, compatibility between them is useful but not vital; according to this camp, efforts to reduce incompatibility should not hinder attempts to improve each language in isolation. The official rationale for the 1999 C standard "endorse[d] the principle of maintaining the largest common subset" between C and C++ "while maintaining a distinction between them and allowing them to evolve separately", and stated that the authors were "content to let C++ be the big and ambitious language."

One commonly encountered difference is that C allows a void\* pointer to be assigned to any pointer type without a cast, whereas C++ does not; this idiom appears often in C code using malloc memory allocation. For example, the following is valid in C but not C++:

Construct that behave differently in c and c++

There are a few syntactical constructs that are valid in both C and C++, but produce different results in the two languages.

For example, character literals such as 'a' are of type int in C and of type char in C++, which means that sizeof 'a' will generally give different results in the two languages: in C++, it will be 1, while in C it will be sizeof(int) which on architectures with 8 bit wide char will be at least 2. As another consequence of this type difference, in C, 'a' will always be a signed expression, regardless of whether or not char is a signed or unsigned type, whereas for C++ this is compiler implementation specific.

C++ implicitly treats any const global as file scope unless it is explicitly declared extern unlike C in which extern is the default. Conversely,  functionss in C are of file scope whereas they have external linkage by default in C++.

**Why to use C ?**

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

* Operating Systems
* Language Compilers
* Assemblers
* Text Editors
* Print Spoolers
* Network Drivers
* Modern Programs
* Data Bases
* Language Interpreters
* Utilities

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient compilers are available for machines ranging in power from the [**Apple**](http://www.apple.com/) Macintosh to the [**Cray**](http://www.cray.com/) supercomputers. The largest measure of C's success seems to be based on purely practical considerations:

1. the portability of the compiler;
2. the standard library concept;
3. a powerful and varied repertoire of operators;
4. an elegant syntax;
5. ready access to the hardware when needed;
6. and the ease with which applications can be optimized by hand-coding isolated procedures

C is often called a "Middle Level" programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (e.g. Fortran) provides everything the programmer might want to do already built into the language. A low level language (e.g. [**assembler**](http://www.le.ac.uk/cc/glossary/ccgla.html#36)) provides nothing other than access to the machines basic instruction set. A middle level language, such as C, probably doesn't supply all the constructs found in high-languages - but it provides you with all the building blocks that you will need to produce the results you want!

**C Compilers**

When you write any program in C language then to run that program you need to compile that program using a C Compiler which converts your program into a language understandable by a computer. This is called machine language (ie. binary format).

**Function**

A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, function strcat() to concatenate two strings, function memcpy() to copy one memory location to another location and many more functions.

A function is known with various names like a method or a sub-routine or a procedure, etc.

## Defining a Function:

* **Return Type**: A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
* **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
* **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
* **Function Body:** The function body contains a collection of statements that define what the function does.

[**How to Win at Tic-Tac-Toe**](file:///H:\CU%20Study\Tic%20Tac%20Toe%20game\How%20to%20Win%20at%20Tic%20Tac%20Toe%20(with%20Strategy%20Examples)_files\How%20to%20Win%20at%20Tic%20Tac%20Toe%20(with%20Strategy%20Examples).htm)

### Method 1 of 2: If You Go First:-

* If you mark the center as your first move, the opponent will then either mark a corner or the edges.
  + If they mark an **edge**, put your next mark in any of the two corners furthest from their mark. If they block your attempt to win, you are on your way to winning! Just simply block their own attempt to win by placing your mark in that particular corner and you will be in a position where you will have **two** chances of winning.
  + If they mark a **corner**, you should place your mark in such a way that all three marks will form a straight diagonal line. If their next move is on an edge, you will have **two** chances of winning.
* If you mark a corner square as your first move, there will be two potential responses by your opponent.
  + If they chose to mark the other squares, you can win. Find the square in which you made your first move. There should be a horizontal row and a vertical row of squares that contain it. Pick a row that doesn’t contain your opponent’s move.
  + Now it’s time to trick your opponent. Look to see if you can win immediately or if you should counter-attack in the center. If you can win right now, make the winning move! If you need to go in the center, then do so and create a situation where you can win in two different places.
  + If they mark the center, you will mark in such a way that all the three marks will form a straight diagonal line.
  + If they **(a)** mark a corner, you are well on the way to winning! Mark the last corner. You will have two chances of winning. However, if they **(b)** mark any other squares, the best scenario is a tie, assuming you keep counter-attacking.

### Method 2 of 2: If Your Opponent Goes First

* If they mark the center, you must mark a corner immediately.
  + If they place their mark in the opposite corner of yours, they are actually using the strategy above! Put your next mark in any square that's not next to your first mark.
  + If they mark any other squares, simply block them. You will end up in a tie.
* If they mark a corner, you must mark the center or you will lose against a good opponent.
  + After you mark the center, they have two choices.
    - If they place their mark such that all the marks will form a straight diagonal line, mark any edges and avoid marking any cornes. You will end up in a tie.
    - Otherwise, just counter-attack their moves and you will have a tie.

### If wondering where would be a safe move in any Tic Tac Toe situation, you first try making a move that will result in a win for you, or if that won't work, block your opponent from winning.

**Possible Problems**

1.) The third player comes: the server will tell him to wait

2.) Wrong turn: the servers will response

3.) Illegal move: the servers will response.

4.) The game is tied: we provide a special method to calculate whether the game is tied, if it is, we start a new game.

5.) Player disconnect: if one disconnect, the server disconnects both players.

**Skill Development**

The most important thing is to never stop learning. I think one should always try to make a conscious effort to improve.

I believe on “learn one programming language every year” system as each programming language changes the way we think about programming.

Looking back at the code you have written in the past and realizing just how bad it was. As Churchill says, “Those that fail to learn from history are doomed to repeat it.”

Pair programming with other programmers might increase the quality of code, broadened your horizons, and help you work in as a team member.

Reading and figuring out code written by different people makes you differently.

Write documentation for code written by other people. This helps in understanding the way other people think and program.

Explaining something really helps you find the gaps in your own knowledge..

My skills which have been improved are:

* Self confidence
* Better hold in programming
* Communication skills

Tic-Tac-Toe is an interactive online version of the classic paper-and-pen game. Kids will learn to apply their logic and form a strategy, as they try to place their symbol -three in a row, to win the game. They will enjoy the challenge as they play against the computer as their play pal and enhance their logical and reasoning skills. This game helps develop coordination, fine motor skills and visual skills in children. The game sharpens the mind and helps to increase the concentration level of the player.

Tic-Tac -Toe program help us to do improve our skill in Function and source code. It helps us to improve our logical skill also.

**FUTURE SCOPE**

Blogging, as you might have heard, is changing the face of the media. It may also be changing the face of mathematical research. For the first time ever, a substantial mathematical problem has been solved via an accumulation of blog comments, all building on each other. Could this be the future of mathematical research?

Before I explain the problem, let’s talk a little about tic-tac-toe. As you probably figured out long ago, intelligent players of ordinary tic-tac-toe (on a 3 by 3 board) will invariably battle to a draw. But, as you probably also figured out, not every game ends in a draw, because not every player is intelligent.

Tic-tac-toe in three dimensions, with three ordinary boards stacked on top of each other (giving you a total of 27 places to place your X.) Now there are 81 squares, and if you want to prevent any possibility of winning, you’re going to have to blacken at least 29 of them. In 5 dimensions, you’ve got to blacken at least 93.

**Flow Chart**

Flowchart is a diagrammatic representation of an algorithm. Flowchart are very helpful in writing program and explaining program to others.

## Symbols Used In Flowchart

Different symbols are used for different states in flowchart, For example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

| Symbol | Purpose | Description |
| --- | --- | --- |
| Flowline symbol in flowchart of programming | Flow line | Used to indicate the flow of logic by connecting symbols. |
| Terminal symbol in flowchart of programming | Terminal(Stop/Start) | Used to represent start and end of flowchart. |
| Input/Output symbol in flowchart of programming | Input/Output | Used for input and output operation. |
| Processing symbol in flowchart of programming | Processing | Used for airthmetic operations and data-manipulations. |
| Decision making symbol in flowchart of programming | Desicion | Used to represent the operation in which there are two alternatives, true and false. |
| On-page connector symbol in flowchart of programming | On-page Connector | Used to join different flowline |
| Off-page connector symbol in flowchart of programming | Off-page Connector | Used to connect flowchart portion on different page. |
| Predefined process symbol in flowchart of programming | Predefined Process/Function | Used to represent a group of statements performing one processing task. |

**Flow Chart**

User

YES

YES

NO

If 2 win

END OF GAME

If 1 win

Match Draw

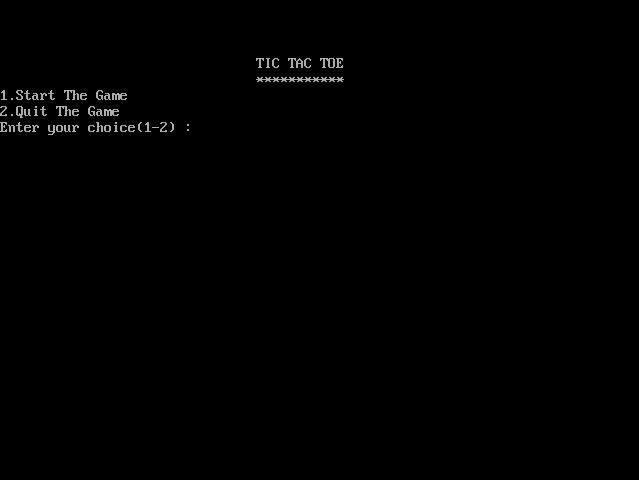
Turn of User 1

Turn of User 2

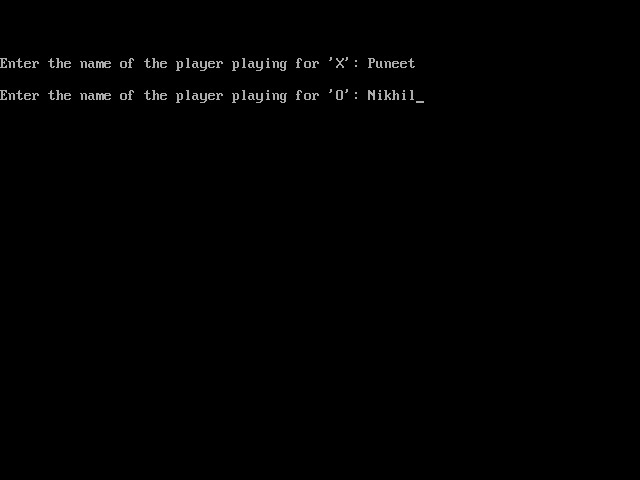
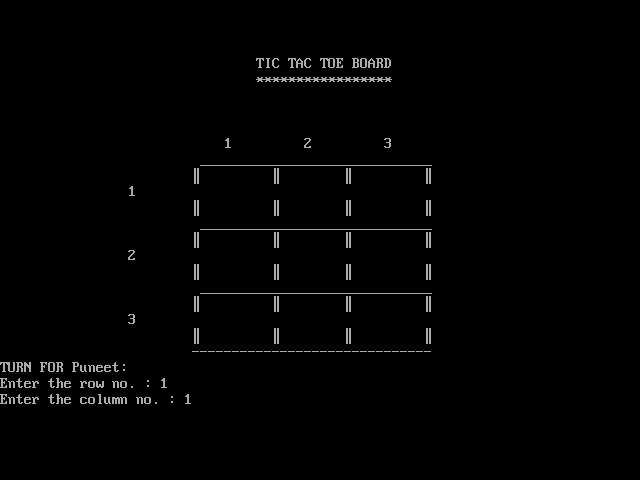
Draw Board

Login of user name etc.

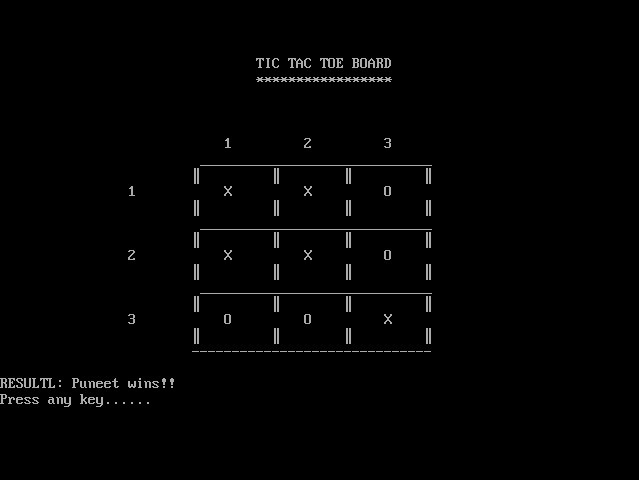
**Screen shots of Output**

****

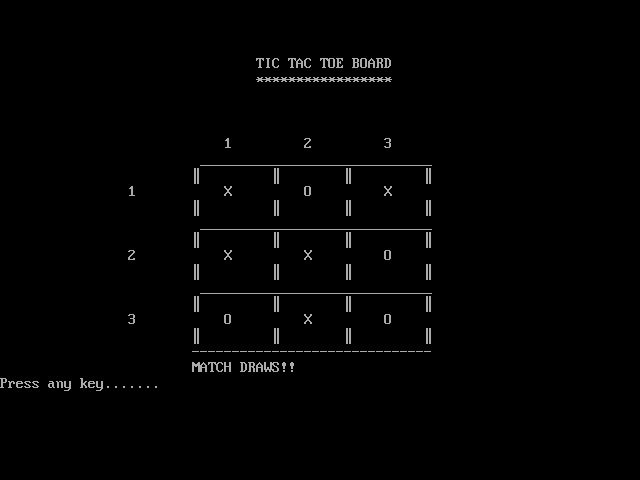
(The picture is showing the starting page of game)

(This picture shows us the initialization of the names of two players)

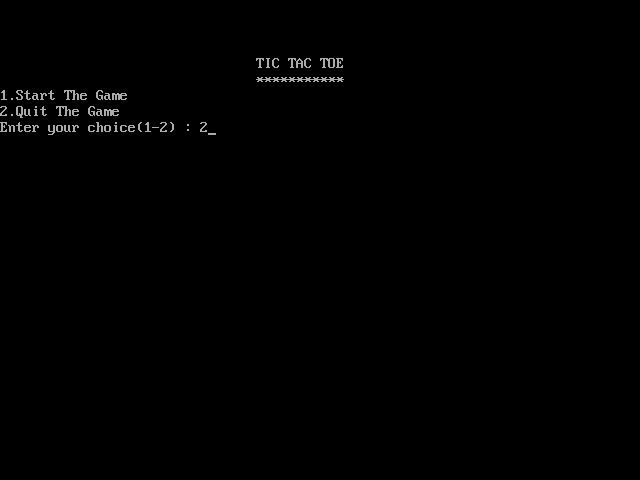
(Board of the game)



(In this picture player1 wins with his various moves by putting the Xs in row and column.)



(This shows us Match Draw condition)



(To quit the game)



(End Page)

**Project and Activity**

You have probably played the Tic-Tac-Toe game to pass time during school hours. It’s fun when you play with paper and pencil. Here, we have developed a console application without graphic for this game as a mini project in C Tic-Tac-Toe game. It is the same noughts and crosses or the Xs and Os, the other names for Tic-Tac-Toe, you played with paper and pencil. This project is compiled in Code::Blocks with gecko compiler.

You can play the game choosing either X or O. If you are playing with X and you input 2, the X will go to first row – second column. If you want to place X in third row – first column, you have to enter 7. And, it is similar for the other positions. This mini project is complete and totally error-free.

Tic-tac-toe is not a very challenging game for human beings. If you’re an enthusiast, you’ve probably moved from the basic game to some variant like three-dimensional tic-tac-toe on a larger grid. If you sit down right now to play ordinary three-by-three tic-tac-toe with a friend, what will probably happen is that every game will come out a tie. Both you and your friend can probably play perfectly, never making a mistake that would allow your opponent to win.

**Results and Discussions**

To my thinking, there are 1145 possible games of tic-tac-toe, since I am reluctant to assume that a player will always see and execute a forking move if possible, but I think that it is reasonable to assume that anyone who can play and understand what they are doing will block when possible. I don't think this is the best answer to the question, though. I think it's better to stick to the rules of the game and assume that the players may be stupid.

I do think that the proper way to count the number of games is to consider the decisions faced by the players. Therefore, the best answer to the question is probably 26,830. This is what you get if you don't end the game until three in a row is completed. I think it's also fair to consider the game to be over when the outcome is fully determined. With the players free to act as stupidly as they wish, this still cuts out some meaningless paths, and gives an answer.

**CONCLUSION**

My original goal was to create a Tic-Tac-Toe game the users play with their hands. I did not meet this goal, however I got pretty close. I got All of the computer vision aspects of my project done, and all that is left is translating mat lab code into C(not a lot though, just if-else statements), programming a tic-tac-toe game (3 hours of work max), and manually translating bytes into useful information (gush lots of work). The big set-back in my project was how difficult it was.

If I had wanted to I could have, as the Prof said "gotten away with murder" in terms of detecting where in a 3x3 grid someone’s hands were, and then based on whose turn it was say it was an X or an O. But instead of focusing on the tic-tac-toe game part of the project, I focused on the computer vision aspects, and felt like I learned a lot more that way and have a better final product.

**Bibliography**

* en.wikipedia.org/wiki/C\_(programming\_language)
* [www.roseindia.net](http://www.roseindia.net)
* www.cprogramming.com/
* www.tutorialspoint.com/cprogramming/

# Let Us C Solutions by [Yashavant P. Kanetkar](http://www.amazon.in/s/280-8487954-9311234?_encoding=UTF8&field-author=Yashavant%20P.%20Kanetkar&search-alias=stripbooks)

* Programming in ANSI C by [E. Balagurusamy](http://www.amazon.in/s?_encoding=UTF8&field-author=E.%20Balagurusamy&search-alias=stripbooks)
* [www.csepidea.com](http://www.csepidea.com)
* www. Ibm.com
* [www.intel.com/India](http://www.intel.com/India)
* A. Stevens, C++ Database Development
* J. Martin, Computer Data-base Organization