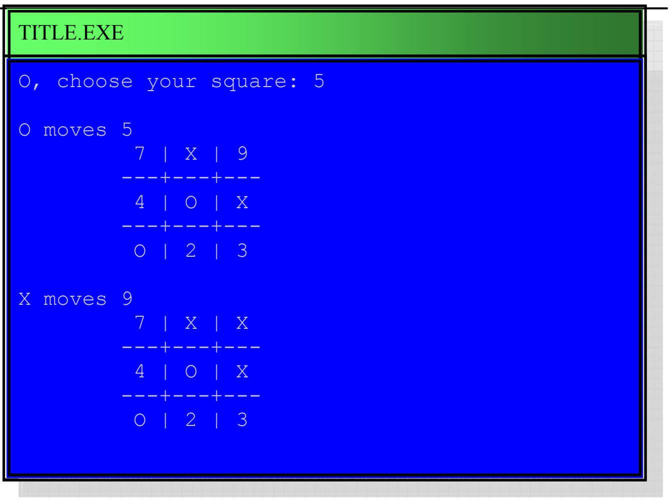
Tic-Tac-Toe



```
TITLE.EXE

O, choose your square: 5

O moves 5
         7 | X | 9
        ---+---+---
         4 | O | X
        ---+---+---
         O | 2 | 3

X moves 9
         7 | X | X
        ---+---+---
         4 | O | X
        ---+---+---
         O | 2 | 3
```

Tic-Tac-Toe is a game that needs no introduction. At some point everyone's learned a winning strategy for the game. What is learned is that if played properly O will never win at Tic-Tac-Toe, but if O plays properly, neither will X. As a computer game, however, this strategy isn't much fun. Who wants to play a computer if you can never beat it, or worse, if you can find and exploit a bug in the game every time.

This version of Tic-Tac-Toe isn't like that. This version starts as a clean slate, like you the first time you learned, with only the basic rules of (1) take the win if you can and (2) block if you have to. Every game it plays it remembers and it remembers how the game ended. Even if you beat it when you were X, when it gets a turn to play X you may begin to see your own moves used on you. If the computer starts winning you've got no one to blame but yourself.

Tic-Tac-Toe was written by Joseph Larson

TicTacToe.CPP	You will need: a C/C++ complier .

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include <time.h>
using namespace std;

#define MAX 1000
#define REM(X) if (turn++ < 7) memory[memory[0]] = memory[memory[0]] * 10 + X
#define XorO(p) (char)((p == 1) ? 'X' : 'O')

int board[9];
int turn;
long memory[MAX];
int stats[2][3]; // [allgames/thisgame][X/O/Tie, win/lose/draw];
int inarow[8][3] = {
  {0,1,2} ,{3,4,5} ,{6,7,8} ,{0,4,8}, {0,3,6} ,{1,4,7} ,{2,5,8} ,{2,4,6}
};

void init (void);
int playgame (int nump);
void drawboard (void);
char XO (int square);
void drawline (int start);
int win (void);
int compymove (int player);
int playermove (int player);
int playagain (void);
void savememory (void);

int main (void) {
  int numplayers;

  init ();
  cout << "tic-tac-toe\n-----------\n";
  cout << "This game of tic-tac-toe is designed to learn for past games 
and\n"
    << "play better as it progresses.\n"
```

Listing continued on page 2...

TicTacToe.CPP	Listing continued from 1...

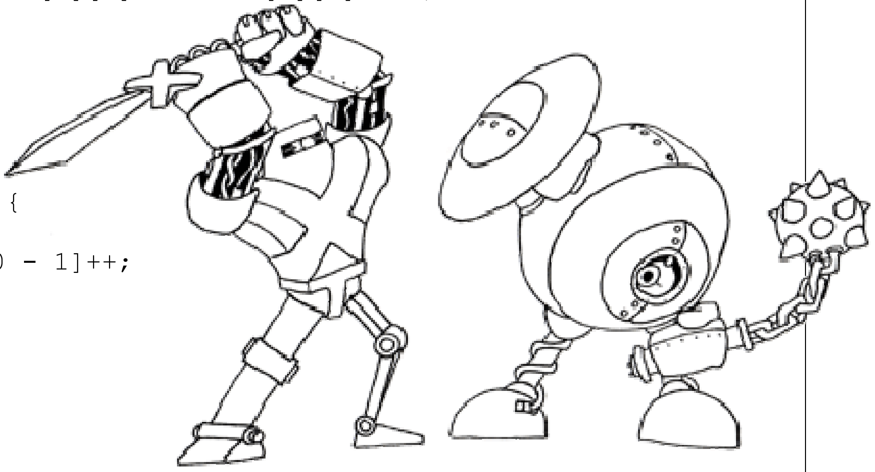
```
    << "See how long it takes before it stops losing...\n\n"
    << "How many human players ? (1-2) : ";
  cin >> numplayers;
  while (playgame (numplayers));
  savememory ();
  return 0;
}

void init (void) {
  ifstream datafile;
  int c;

  for (c = 0; c < 3; c++) stats[0][c] = stats[1][c] = 0;
  srand (time (NULL));
  c = 1;
  datafile.open ("ttt.dat");
  if (!datafile.is_open())
    memory[0] = 1;
  else {
    while (!datafile.eof ()) {
      datafile >> memory[c];
      stats[0][memory[c] % 10 - 1]++;
      c++;
    }
    datafile.close();
    memory[0] = --c;
  }
}

int playgame (int nump) {
  int c, move, player = 1, human[3];

  for (int c = 0; c < 9; c++) board[c] = 0;
  memory[memory[0]] = 0;
  turn = 0;
  for (c = 0; c < 2; c++) human[c] = 1; human [2] = 3;
  c = rand () < (RAND_MAX / 2);
  while (nump-- > 0) {
    human[c++] = 2;
    cout << "Human plays " << XorO(c) << "\n";
    c %= 2;
  }
  while (!win()) {
    drawboard ();
    move = (human[player - 1] == 2) ? playermove (player) : compymove
(player);
    cout << "\n" << XorO(player) << " moves " << move + 1 << "\n";
    board[move] = player;
    REM (move);
    player %= 2; player++;
  }
  drawboard ();
  move = win ();
  switch (move) {
    case 1 : cout << "\nX wins!\n"; break;
    case 2 : cout << "\nO wins!\n"; break;
    case 3 : cout << "\nTie game\n"; break;
  }
  stats[1][human[move - 1] - 1]++;
```



Listing continued on page 3...

Continued from page 2...

```
  stats[0][move - 1]++; // Update total stats.
  while (turn < 9) REM (0);
  memory[memory[0]] = memory[memory[0]] * 10 + move; // Store winner.
  if (++memory[0] > MAX) {
    for (c = 1; c < MAX; c++) memory[c] = memory[c + 1];
    memory[0]=MAX;
  }
  cout << "\nHuman Stats:\n" << "Wins : " << stats[1][1]
  << "\tLoses : " << stats[1][0] << "\tDraws : " << stats[1][2]
  << "\n\nAll Games:\n" << "X : "  << stats[0][0] << "\tO : " << stats[0][1]
  << "\tDraw : " << stats[0][2] << "\tTotal Games : " << memory[0] - 1;
  return (playagain ());
}

void drawboard (void) {
  cout << "\t " << XO(6) << " | " << XO(7) << " | " << XO(8) << '\n';
  cout << "\t---+---+---\n";
  cout << "\t " << XO(3) << " | " << XO(4) << " | " << XO(5) << '\n';
  cout << "\t---+---+---\n";
  cout << "\t " << XO(0) << " | " << XO(1) << " | " << XO(2) << '\n';

}

char XO (int square) {
  switch (board[square]) {
    case 0 : return (square + '1');
    case 1 : return 'X';
    case 2 : return 'O';
  }
}

int playermove(int player) {
  int move=-1;

  do {
    if (move >= 0) cout << "Choose an empty square.\n";
    cout << "\n" << XorO(player) << ", choose your square: ";
    cin >> move;
  } while (move < 1 || move > 9 || board[--move]);
  return move;
}

int compymove (int player) {
  int move, c, check, rank[9];

  for (c = 0; c < 9; c++) rank [c] = MAX;
  for (move = 0; move < 9; move++) { // look for a three in a row to win.
    if (!board[move]) {
      board[move] = player;
      if (win()) return move;
      board[move] = 0;
    }
  }
  for (move = 0; move < 9; move++) { // look for a three in a row to block.
    if (!board[move]) {
      board[move] = player % 2 + 1;
      if (win()) return move;
      board[move] = 0;
    }
  }
```

Listing continued on page 2...

TicTacToe.CPP	Listing Continued from page 1….

```
  for (move = 0; move < 9; move++) { // rank the moves according to memory.
    if (turn < 7) check = memory[memory[0]] * 10 + move;
    for (c = 1; c < memory[0]; c++)
      if (check == (int)(memory[c] / pow(10, 7 - turn)))
        if (memory[c] % 10 == player) rank[move]++;
        else if (memory[c] % 10 == player % 2 + 1) rank[move]--;
  }
  c = 0; check = 0;
  for (move = 0; move < 9; move ++) // How many moves have the high rank?
    if (!board[move])
      if (!c) {c = 1; check = rank[move];}
      else if (rank[move] == check) c++;
      else if (rank[move] > check) {c = 1; check = rank[move];}
  c -= rand () % c; // pick the 'c'th high ranker.
  for (move = 0; move < 9; move++) {
    if (!board[move] && rank[move] == check) c--;
    if (!c) return move;
  }
}

int win (void) {
  if (turn > 8) return 3;
  for (int c = 0; c < 8; c++)
    if ((board[inarow[c][0]] != 0) &&
       (board[inarow[c][0]] == board[inarow[c][1]]) &&
       (board[inarow[c][0]] == board[inarow[c][2]]))
        return board[inarow[c][0]];
  return 0;
}

int playagain (void) {
  char yesno;

  cout << "\n\nI play better the more you play.\n"
    << "Do you want to play again?";
  cin >> yesno;
  if (yesno == 'y' || yesno == 'Y') return 1;
  else return 0;
}

void savememory (void) {
  ofstream datafile ("ttt.dat");
  int c;

  if (datafile.is_open ()) {
    for (c = 1; c <= memory[0] - 1; c++) datafile << memory[c] << '\n';
    datafile.close();
  } else cout << "Can not open file. Memory not saved.";
}
```

Author's Notes:

The initial version of this program allowed for 1 or 2 human players, but I rewrote it later to play itself if you enter 0 for number of players. It's interesting to watch the computer build up a library of winning moves by playing itself.

Potentially having the computer check for wins and blocks could be removed and it would still eventually play well, but it would require a significantly bigger memory array. Still, for academic purposes it may be fun to build a version that had no knowledge of the rules and see how long it would take playing itself before it built up a winning memory.