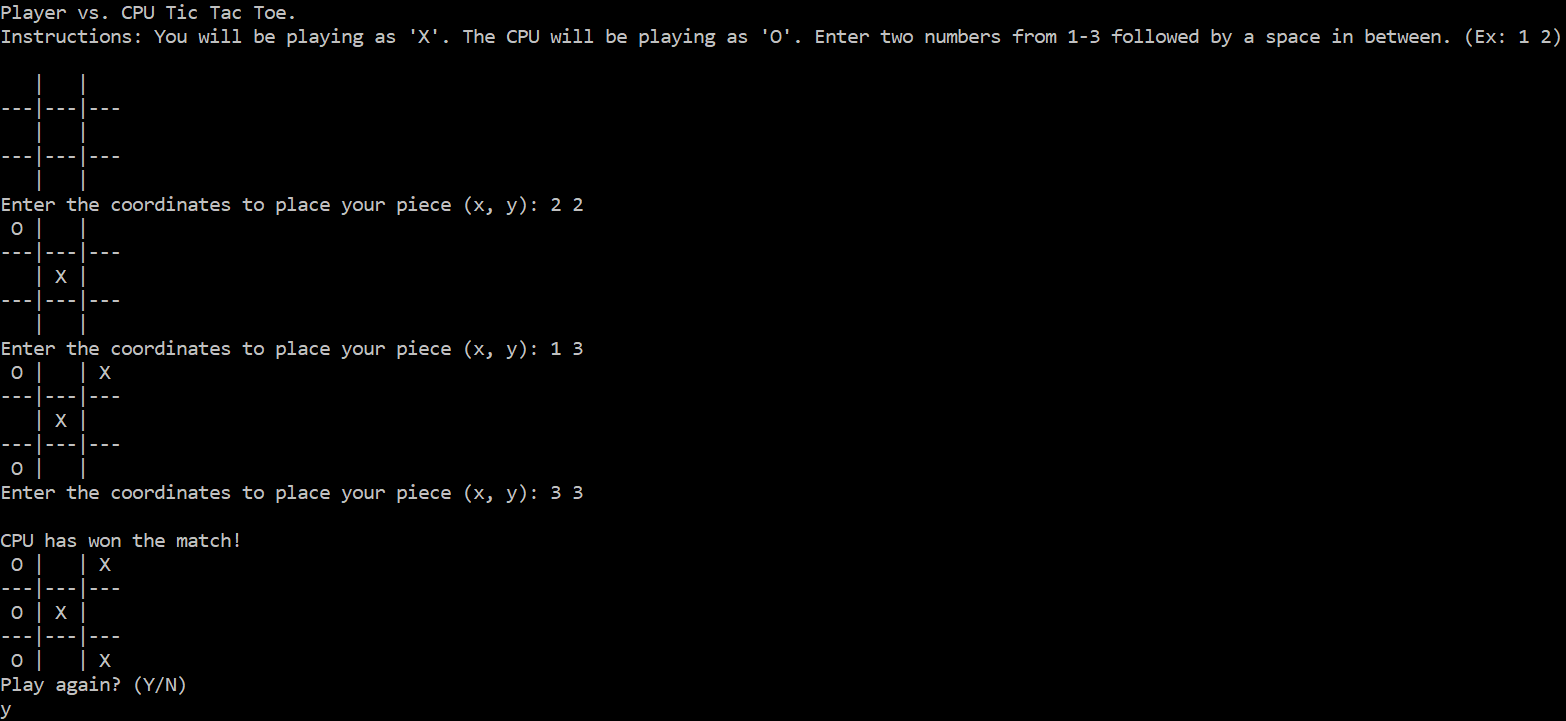
Calvin Truong

Tic-Tac-Toe Game Project

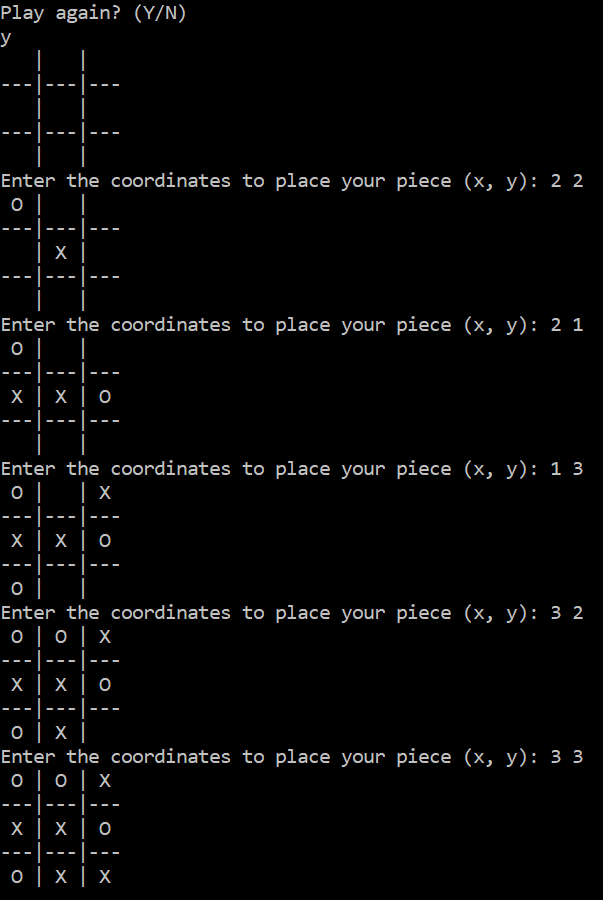
3/25/18

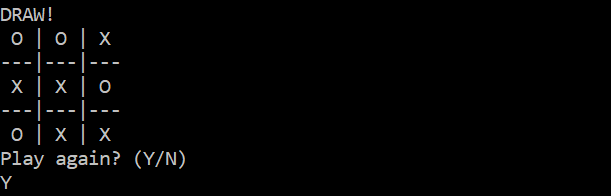
**Output Results:**

Game 1

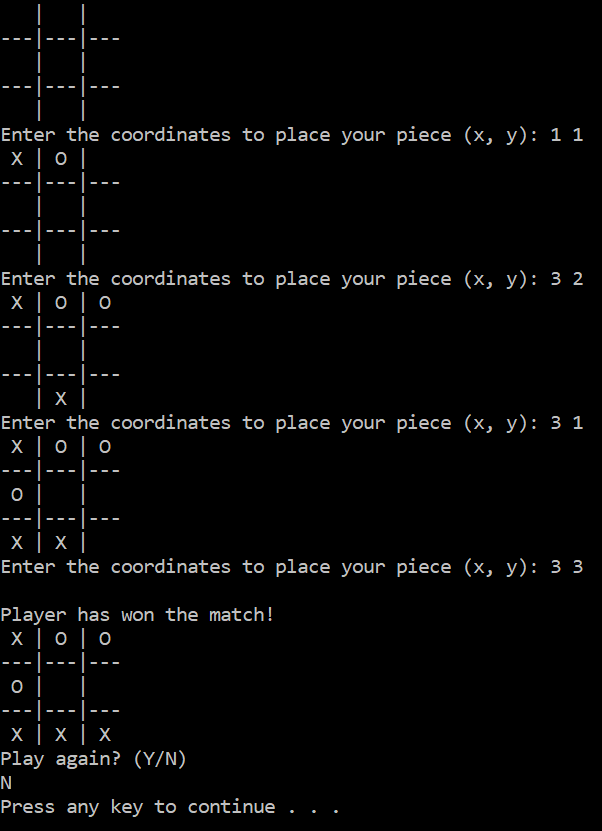


Game 2





Game 3



// Tic\_Tac\_Toe\_Project.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include <iostream>

using namespace std;

int row, column, diagonal;

int matrix[3][3]; // create a 3x3 matrix

int emptyrow(int c);

int emptycol(int r);

void init\_matrix(void); // initialize matrix

void get\_player\_move(void);

void get\_computer\_move(void);

void display\_matrix(void);

char check(void);

bool vulnerable(int mat[3][3]);

bool winnable(int mat[3][3]);

bool finish(int mat[3][3]);

void restart();

int main(void)

{

char done, c;

bool CONTINUE = true;

cout << "Player vs. CPU Tic Tac Toe.\n";

cout << "Instructions: You will be playing as 'X'. The CPU will be playing as 'O'. Enter two numbers from 1-3 followed by a space";

cout << " in between. (Ex: 1 2) \n" << endl;

while (CONTINUE)

{

done = ' ';

init\_matrix();

do {

display\_matrix();

get\_player\_move();

done = check(); // Checks for potential winner

if (done != ' ')

{

break; // when winner is found

}

get\_computer\_move();

done = check(); // Checks for winner again

if (done != ' ')

{

break; // when winner is found

}

if (finish(matrix)) // winner is not found and there are no open spaces left.

{

display\_matrix();

cout << endl;

cout << "DRAW!\n";

break;

}

restart();

} while (done == ' ');

if (done == 'X')

{

cout << "\nPlayer has won the match! \n";

}

if (done == 'O')

{

cout << "\nCPU has won the match! \n";

}

display\_matrix(); // shows the final matrix display

cout << "Play again? (Y/N)" << endl;

cin >> c;

if (c == 'N' || c == 'n')

{

CONTINUE = false;

}

restart();

}

return 0;

}

int emptyrow(int c)

{

for (int i = 0; i < 3; ++i)

{

if (matrix[i][c] == ' ')

{

return i;

}

}

}

int emptycol(int r)

{

for (int i = 0; i < 3; ++i)

{

if (matrix[r][i] == ' ')

{

return i;

}

}

}

void init\_matrix(void)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

matrix[i][j] = ' ';

}

}

}

void get\_player\_move(void) // calls the function for user's input

{

int x, y;

cout << "Enter the coordinates to place your piece (x, y): ";

cin >> x >> y;

x--; y--;

if (matrix[x][y] != ' ')

{

cout << "Unable to read, try again. \n";

get\_player\_move();

}

else matrix[x][y] = 'X';

}

void get\_computer\_move(void)

{

char temporary;

int test[3][3], temp[3][3];

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 3; ++j)

{

test[i][j] = matrix[i][j];

temp[i][j] = matrix[i][j];

}

}

int i, j;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (matrix[i][j] == ' ')

{

temporary = matrix[i][j];

if (winnable(test))

{

if (row > 0)

{

if (matrix[row - 1][emptycol(row - 1)] != 'X')

matrix[row - 1][emptycol(row - 1)] = 'O';

return;

}

if (column > 0)

{

if (matrix[emptyrow(column - 1)][column - 1] != 'X')

matrix[emptyrow(column - 1)][column - 1] = 'O';

return;

}

if (diagonal == 1)

{

if (matrix[0][0] != 'X' && matrix[1][1] != 'X' && matrix[2][2] != 'X')

matrix[0][0] = 'O'; matrix[1][1] = 'O'; matrix[2][2] = 'O';

return;

}

else

{

if (matrix[0][2] != 'X' && matrix[1][1] != 'X' && matrix[2][0] != 'X')

matrix[0][2] = 'O'; matrix[1][1] = 'O'; matrix[2][0] = 'O';

return;

}

return; // break;

}

test[i][j] = temporary;

}

}

if (matrix[i][j] == ' ')

{

temporary = matrix[i][j];

if (winnable(test))

{

if (row > 0)

{

if (matrix[row - 1][emptycol(row - 1)] != 'X')

matrix[row - 1][emptycol(row - 1)] = 'O';

return;

}

if (column > 0)

{

if (matrix[emptyrow(column - 1)][column - 1] != 'X')

matrix[emptyrow(column - 1)][column - 1] = 'O';

return;

}

if (diagonal == 1)

{

if (matrix[0][0] != 'X' && matrix[1][1] != 'X' && matrix[2][2] != 'X')

matrix[0][0] = 'O'; matrix[1][1] = 'O'; matrix[2][2] = 'O';

return;

}

else

{

if (matrix[0][2] != 'X' && matrix[1][1] != 'X' && matrix[2][0] != 'X')

matrix[0][2] = 'O'; matrix[1][1] = 'O'; matrix[2][0] = 'O';

return;

}

return; // break

}

test[i][j] = temporary;

}

}

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (matrix[i][j] == ' ')

{

temporary = matrix[i][j];

test[i][j] = 'O';

if (!vulnerable(test))

{

break;

}

test[i][j] = temporary;

}

}

if (matrix[i][j] == ' ')

{

temporary = matrix[i][j];

test[i][j] = 'O';

if (!vulnerable(test))

{

break;

}

test[i][j] = temporary;

}

}

if (column > 0)

{

if (matrix[emptyrow(column - 1)][column - 1] != 'X')

matrix[emptyrow(column - 1)][column - 1] = 'O';

return;

}

if (row > 0)

{

if (matrix[row - 1][emptycol(row - 1)] != 'X')

matrix[row - 1][emptycol(row - 1)] = 'O';

return;

}

if (diagonal == 1)

{

if (matrix[0][0] != 'X')

matrix[0][0] = 'O';

if (matrix[1][1] != 'X')

matrix[1][1] = 'O';

if (matrix[2][2] != 'X')

matrix[2][2] = 'O';

return;

}

if (diagonal == 2)

{

if (matrix[0][2] != 'X')

matrix[0][2] = 'O';

if (matrix[1][1] != 'X')

matrix[1][1] = 'O';

if (matrix[2][0] != 'X')

matrix[2][0] = 'O';

return;

}

if (matrix[i][j] != 'X')

matrix[i][j] = 'O';

}

void display\_matrix(void) // displays the matrix

{

int d;

for (d = 0; d < 3; d++)

{

cout << " " << char(matrix[d][0]) << " | " << char(matrix[d][1]) << " | " << char(matrix[d][2]);

if (d != 2)

{

cout << "\n---|---|---\n";

}

}

cout << endl;

}

char check(void) // used to determine winner

{

int i;

for (i = 0; i < 3; i++) // Checks the rows

{

if (matrix[i][0] == matrix[i][1] && matrix[i][0] == matrix[i][2])

{

return matrix[i][0];

}

}

for (i = 0; i < 3; i++) // Checks the columns

{

if (matrix[0][i] == matrix[1][i] && matrix[0][i] == matrix[2][i])

{

return matrix[0][i];

}

}

// test the diagonals

if (matrix[0][0] == matrix[1][1] && matrix[1][1] == matrix[2][2])

{

return matrix[0][0];

}

if (matrix[0][2] == matrix[1][1] && matrix[1][1] == matrix[2][0])

{

return matrix[0][2];

}

return ' ';

}

//Looks for all of the possible outcome of the user winning the game

bool vulnerable(int mat[3][3])

{

char a, b, c, d, e, f, g, h, i;

a = mat[0][0], b = mat[0][1], c = mat[0][2];

d = mat[1][0], e = mat[1][1], f = mat[1][2];

g = mat[2][0], h = mat[2][1], i = mat[2][2];

if ((a == 'X' && b == 'X' && c != 'O') || (b == 'X' && c == 'X' && a != 'O') || (a == 'X' && c == 'X' && b != 'O'))

{

row = 1;

return true;

}

if ((d == 'X' && e == 'X' && f != 'O') || (e == 'X' && f == 'X' && d != 'O') || (d == 'X' && f == 'X' && e != 'O'))

{

row = 2;

return true;

}

if ((g == 'X' && h == 'X' && i != 'O') || (h == 'X' && i == 'X' && g != 'O') || (g == 'X' && i == 'X' && h != 'O'))

{

row = 3;

return true;

}

if ((a == 'X' && d == 'X' && g != 'O') || (d == 'X' && g == 'X' && a != 'O') || (a == 'X' && g == 'X' && d != 'O'))

{

column = 1;

return true;

}

if ((b == 'X' && e == 'X' && h != 'O') || (e == 'X' && h == 'X' && b != 'O') || (b == 'X' && h == 'X' && e != 'O'))

{

column = 2;

return true;

}

if ((c == 'X' && f == 'X' && i != 'O') || (f == 'X' && i == 'X' && c != 'O') || (c == 'X' && i == 'X' && f != 'O'))

{

column = 3;

return true;

}

if ((a == 'X' && e == 'X' && i != 'O') || (e == 'X' && i == 'X' && a != 'O') || (a == 'X' && i == 'X' && e != 'O'))

{

diagonal = 1;

return true;

}

if ((g == 'X' && e == 'X' && c != 'O') || (e == 'X' && c == 'X' && g != 'O') || (g == 'X' && c == 'X' && e != 'O'))

{

diagonal = 2;

return true;

}

return false;

}

//Looks for all of the possible outcome of the computer winning the game

bool winnable(int mat[3][3])

{

char a, b, c, d, e, f, g, h, i;

a = mat[0][0], b = mat[0][1], c = mat[0][2];

d = mat[1][0], e = mat[1][1], f = mat[1][2];

g = mat[2][0], h = mat[2][1], i = mat[2][2];

if ((a == 'O' && b == 'O' && c != 'X') || (b == 'O' && c == 'O' && a != 'X') || (a == 'O' && c == 'O' && b != 'X'))

{

row = 1;

return true;

}

if ((d == 'O' && e == 'O' && f != 'X') || (e == 'O' && f == 'O' && d != 'X') || (d == 'O' && f == 'O' && e != 'X'))

{

row = 2;

return true;

}

if ((g == 'O' && h == 'O' && i != 'X') || (h == 'O' && i == 'O' && g != 'X') || (g == 'O' && i == 'O' && h != 'X'))

{

row = 3;

return true;

}

if ((a == 'O' && d == 'O' && g != 'X') || (d == 'O' && g == 'O' && a != 'X') || (a == 'O' && g == 'O' && d != 'X'))

{

column = 1;

return true;

}

if ((b == 'O' && e == 'O' && h != 'X') || (e == 'O' && h == 'O' && b != 'X') || (b == 'O' && h == 'O' && e != 'X'))

{

column = 2;

return true;

}

if ((c == 'O' && f == 'O' && i != 'X') || (f == 'O' && i == 'O' && c != 'X') || (c == 'O' && i == 'O' && f != 'X'))

{

column = 3;

return true;

}

if ((a == 'O' && e == 'O' && i != 'X') || (e == 'O' && i == 'O' && a != 'X') || (a == 'O' && i == 'O' && e != 'X'))

{

diagonal = 1;

return true;

}

if ((g == 'O' && e == 'O' && c != 'X') || (e == 'O' && c == 'O' && g != 'X') || (g == 'O' && c == 'O' && e != 'X'))

{

diagonal = 2;

return true;

}

return false;

}

//Looks for the to see if the spaces are all filled out before ending the game

bool finish(int mat[3][3])

{

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 3; ++j)

{

if (mat[i][j] == ' ')

{

return false;

}

}

}

return true;

}

void restart()

{

row = 0;

column = 0;

diagonal = 0;

}