**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інтелектуальних інформаційних систем**

**ЗВІТ**

**ПРО НАВЧАЛЬНУ ПРАКТИКУ**

Спеціальність 122 «Комп’ютерні науки»

122 – ПРН.ПЗ.00 – 101.31210101

Виконала: студентка 1 курсу, групи 101

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**С. О. Іванова\_

(підпис, ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Керівник: канд. техн. наук, доцент, доцент кафедри ІІС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Є. В. Сіденко

(підпис, ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Комісія:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ І. В. Кулаковська

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. С. Скакодуб

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**м. Миколаїв – 2023 рік**

**ЗМІСТ**

[ВСТУП 3](#__RefHeading___Toc991_31222461)

[1 ЗМІСТОВНА ЧАСТИНА 4](#__RefHeading___Toc993_31222461)

[2 ПРИКЛАДИ ОФОМРЛЕННЯ ЕЛЕМЕНТІВ 5](#__RefHeading___Toc995_31222461)

[ВИСНОВКИ 7](#__RefHeading___Toc997_31222461)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 8

# ВСТУП

Мною були розроблені гра Tic-Tac-Toe, алгоритм для по піксельного виведення елементів масиву з даними гри, алгоритм для пріоретизації пошуку двох однакових елементів по діагоналі, вертикалі та горизонталі в залежності від сторони бота, алгоритм знаходження найкращих перших ходів, навігація між різними режимами гри з використанням regex, алгоритм обнуління пам’яті ботів після кінця раунду, алгоритм аналізу TTT board з метою моніторингу процесу гри та можливість обирати рівень складності гри. Також була реалізована система зациклення гри, яка дозволяє необмежено грати з ботами з можливістю вийти з гри в будь який момент після кінця раунду з можливістю міняти режими гри, сторону гри (‘x’ або ‘o’), складність ботів та сторону гри ботів (‘x’ або ‘o’). У вступі також необхідно вказати назву кафедри або підрозділу, які були місцем проходження навчальної практики та період проходження практики (з 19 вересня по 15 жовтня).

Вступ повинен бути не менше ніж пів сторінки.

# 1 ЗМІСТОВНА ЧАСТИНА

Таблиця 1.1 – завдання:

|  |  |
| --- | --- |
| 1 | Створення Tic-Tac-Toe Board |
| 2 | Введення та виведення даних TTT board |
| 3 | Перевірки ходу гри (x win/o win/ draw/ the game is still on) |
| 4 | Зациклення гри |
| 5 | Псевдографіка (вивід TTT board по пікселям) |
| 6 | Додати бота easy |
| 7 | Додати можливість обирати сторону гри |
| 8 | Додати бота medium |
| 9 | Додати бота hard |
| 10 | Покращення навігації завдяки regex |
| 11 | Фінальне тестування гри |

**Програмні засоби:** Visual studio, Visual studio code, Notepad++

**Мови програмування:** C, C++

**Алгоритми:**

**Алгоритм по піксельного виводу поля:**

Через те що після переходу на нову строку не можливо повернутися назад, необхідно не просто створити 2D моделі ‘x’, ‘o’ і ‘ ’, а й роздробити їх на шари, які потім будуть збиратися в одне єдине поле.

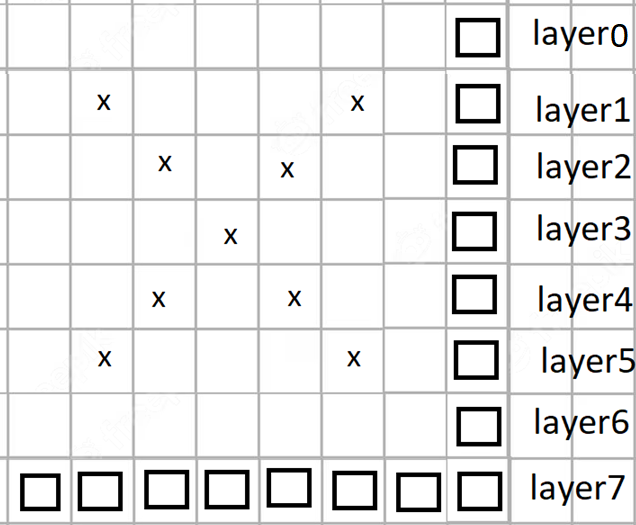


Рисунок 1.1 – шари для виводу ‘x’

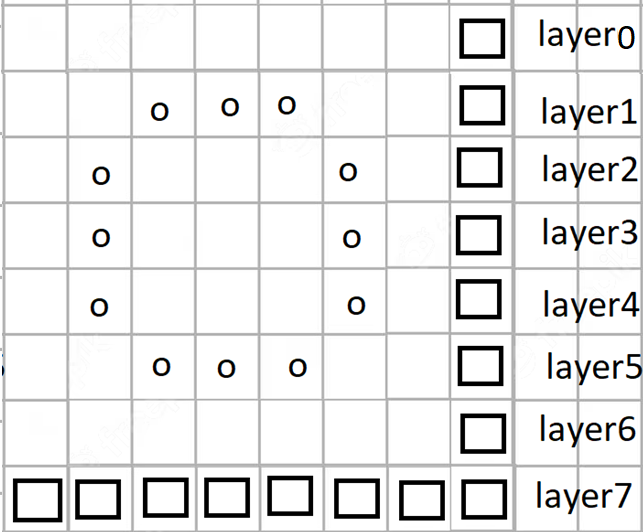


Рисунок 1.2 – шари для виводу ‘о’

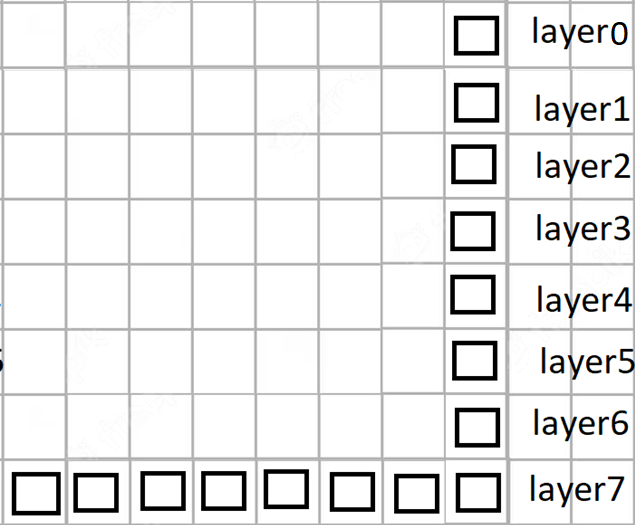


Рисунок 1.3 – шари для виводу ‘ ’

Після цього треба розробити функцію, яка буде збирати шари в одне єдине поле. Для цього необхідно прочитати перші 3 елементи поля і через цикл помістити кожний шар по черзі в масив 8 на 8. Це необхідно проробити 3 рази з вісьма шарами. Розглянемо на прикладі:

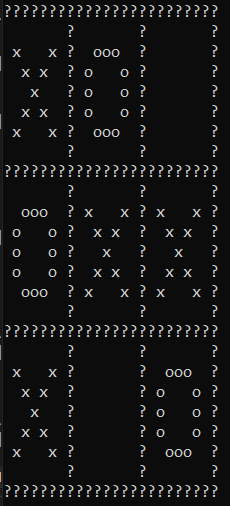


Рисунок 1.4 – приклад дошки

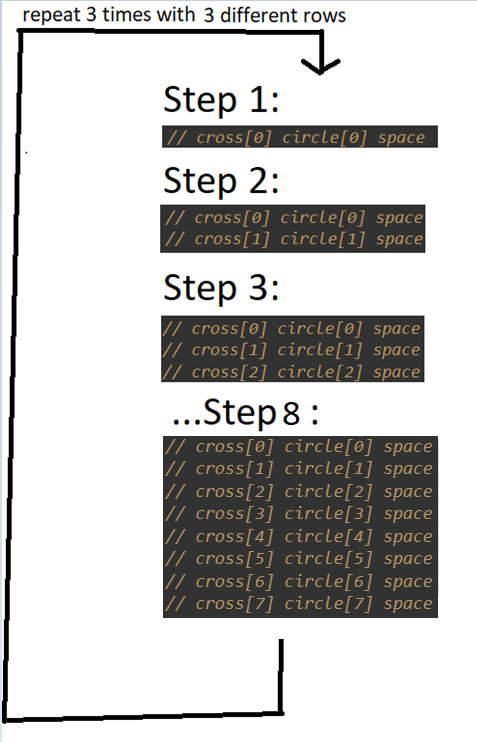


Рисунок 1.5 – алгоритм заповнення поля на основі шарів та самого поля

Після кожної ітерації після збору 3-х елементів (які можуть бути ‘x’/‘o’/‘ ’) їх необхідно вивести на екран і очистити масив 8 на 8.

**Алгоритм botEasy:**

Необхідно створити булевий масив який містить інформацію про те в якій клітинці вже є ‘x’ або ‘o’. Після цього за допомогою функції rand(), беручі до уваги булевий масив, обрати випадкове незайняте місце куди бот поставить ‘x’ або ‘o’.

**Алгоритм botMedium:**

Спочатку необхідно створити шаблони переможних позицій які бот буде порівнювати з полем. Переможні позиції – це позиції, які гарантують перемогу на наступний хід. Одночасно порівнюватися буде тільки один рядок / стовбець. Діагональних перемог набагато менше, тому знаходження цих позицій можна зробити за допомогою **if**ів.

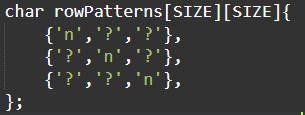


Рисунок 1.6 – Шаблон переможних позицій для рядків та стовбців

Також щоб уникнути неприємних ситуацій необхідно додати боту пріоритет переможних позицій. Тобто у наступній ситуації bot x (бот який грає за x) має обрати комірку “5”, а не комірку “2”

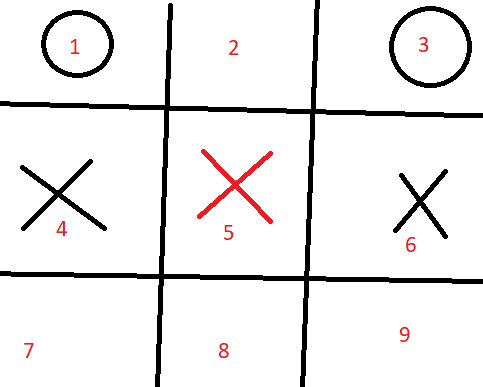


Рисунок 1.7 – приклад роботи пріоритету

Для цього необхідно зробити функцію яка в залежності від сторони бота буде заповнювати rowPatterns відповідно ‘x’ або ‘o’. Наприклад, для bot x, якщо не буде знайдено співпадінь з rowPatters для ‘x’, масив зміниться і бот буде шукати співпадіння з rowPatterns для ‘o’.

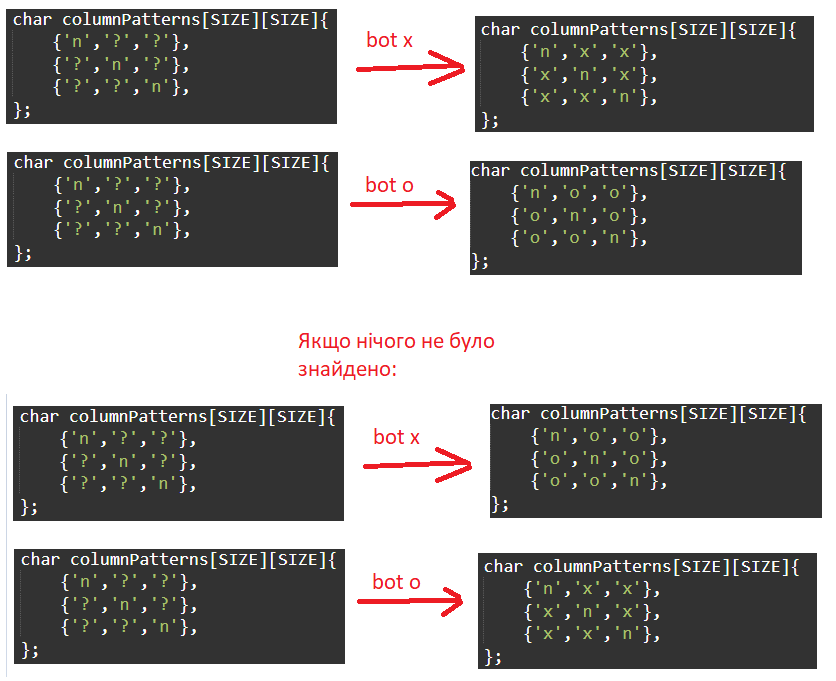


Рисунок 1.8 – зміна пріоритету для ботів в залежності від успіху пошуку

Важливо зауважити, що rowPatterns використовується як для вертикального так і горизонтального пошуку. Діагональний пошук зроблено задяки **if**ів які просто перевіряють наявність 2-х однакових символів (‘x’ або ‘o’) в одній з двух діагоналей. Пріоритет для діагоналі працює аналогічно.

Якщо не було знайдено переможних позицій, викликається botEasy, який ставить ‘x’ або ‘o’ в випадкову вільну клітинку.

**Алгоритм botHard:**

botHard працює ідентично алгоритму botMedium за винятком того, що перші 3 ходи є пре записаними.

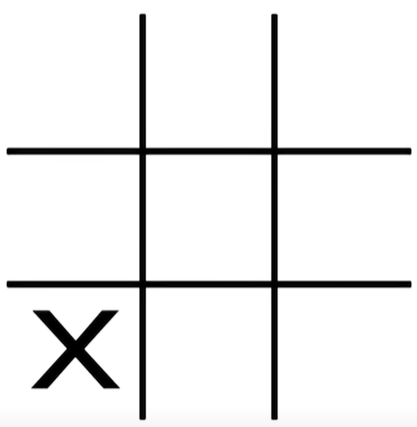


Рисунок 1.9 – перший хід за ‘x’

Якщо людина походить посередні, перемогти майже не можливо. Якщо людина походить в будь яке інше місце, необхідно поставити ‘x’ в протилежний кут.

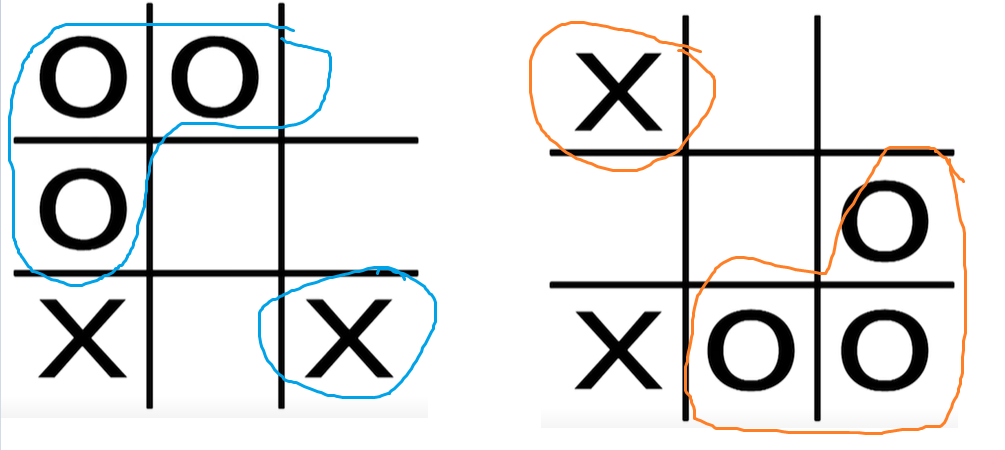


Рисунок 1.10 – другий хід за ‘x’

Після другого ходу необхідно знайти вільне місце яке зможе дати боту одночасно 2 переможні позиції.



Рисунок 1.11 – третій хід за ‘x’

Якщо людина ходить другою, botHard зробить перший хід по центру, а після цього за допомогою botMedium буде блокувати всі переможні позиції гравця.

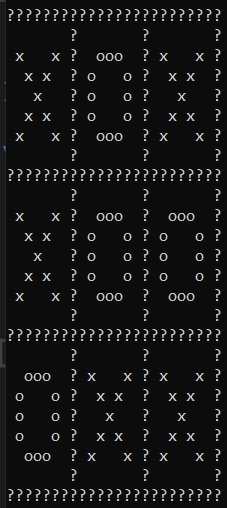


Рисунок 1.12 – приклад гри bot o

Важливо пам’ятати, що перемогти botHard не можливо.

**Алгоритм повторного виклику botHard з використанням статичних зміних:**

Як було зазначено вище, функція botHard працює ідентично функції botMedium за винятком перших трех ходів. Для того, щоб botHard розумів який зараз хід, було створено 3 статичні змінні:

didFirstMoveHappen;

didSecondMoveHappen;

didThirdMoveHappen;

Кожна з цих змінних була прирівняна до нуля:

didFirstMoveHappen = 0;

didSecondMoveHappen = 0;

didThirdMoveHappen = 0;

Також були створенні три функції:

botHardFirstMove, botHardSecondMove, botHardThirdMove

Які відповідають за перші три хода бота. Після виклику першої функції значення зміної didFirstMoveHappen буде прирівняно до 1. Таким чином після повторного виклику функції botHard, botHard зрозуміє, що перший хід вже відбувся і бот перейде до виконання другого ходу після чого аналогічно присвоїть значення didSecondMoveHappen до 1. Аналогічно з didThirdMoveHappen, окрім того, що нема сенсу присвоювати значення didThirdMoveHappen до 1, оскільки після третього ходу весь хід гри перейде в руки botMedium. Якщо Інкрементувати зміну didThirdMoveHappen, це на алгоритм ніяк не вплине.

Основною проблемою є те, що через той факт, що після гри з botHard з ним можна зіграти ще раз, статичні зміні не обнуляються. Для того щоб виправити цю проблему, після кожного виклику функції botHard необхідно перевіряти чи почалася нова гра. Це можна зробити за допомогою функції checkCheckArr яка перевіряє чи всі 9 полів вільні. У такому разі усі статичні змінні будуть обнулінні.

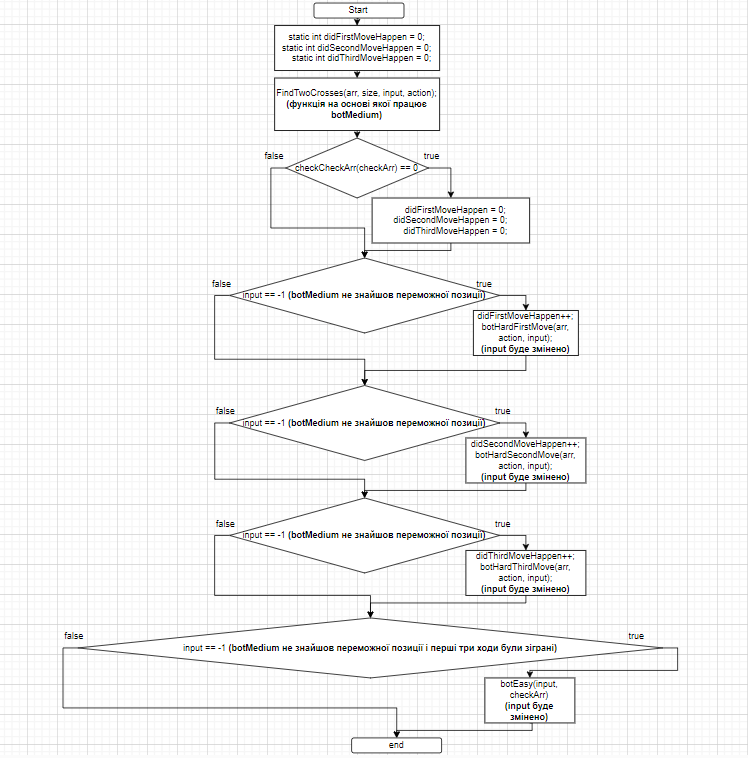


Рисунок 1.13 – блок-схема роботи botHard

**Алгоритм знаходження переможних позицій:**

Цей алгоритм використовується ботами botMedium і botHard. Завдяки масиву з шаблонами переможних позицій (Рисунок 1.6) необхідно порівняти кожний рядок поля гри спочатку з першим рядком масиву шаблонів, потім з другим і третім.

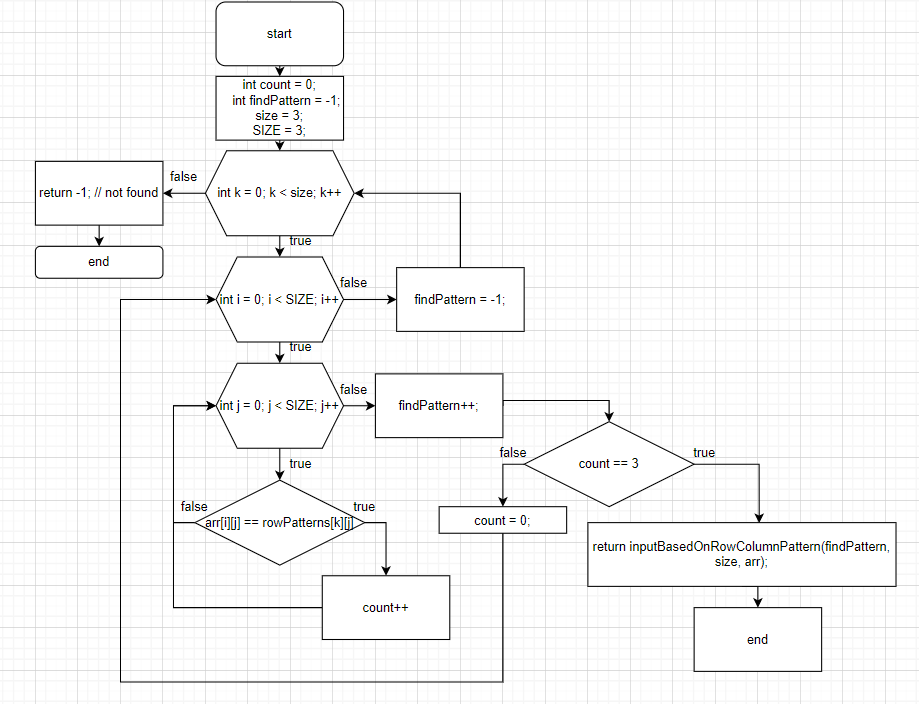


Рисунок 1.14 – блок-схема пошуку горизонтальних переможних позицій

Для знаходження переможних позицій вертикально необхідно виконати ідентичний алгоритм, за винятком того, що arr[i][j] було змінено на arr[j][i].

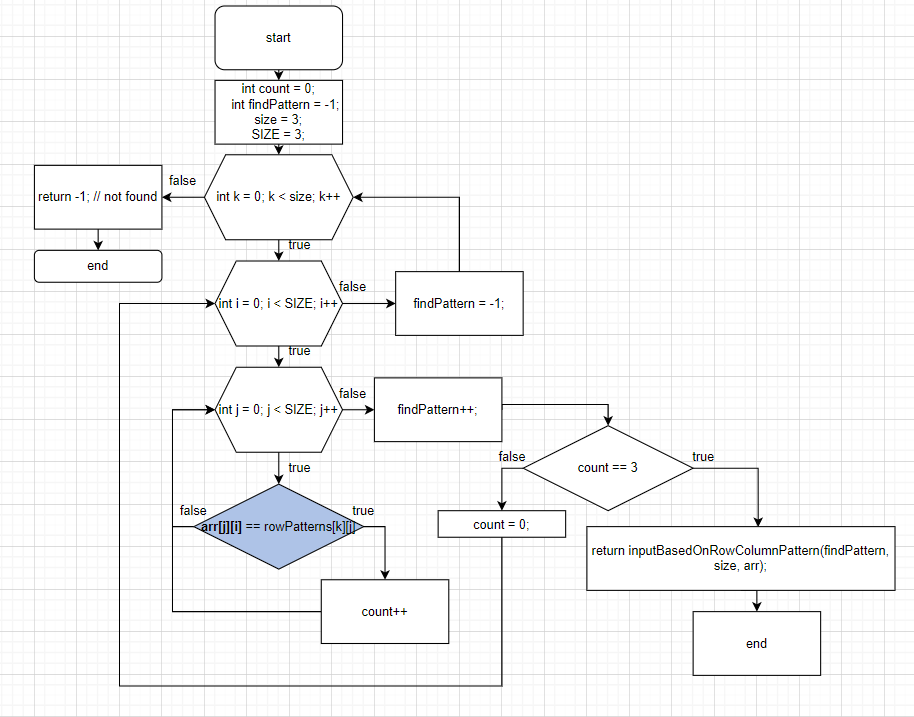


Рисунок 1.15 – блок-схема пошуку вертикальних переможних позицій

Для знаходження діагональних переможних позицій, яких набагато менше ніж горизонтальних і вертикальних достатньо просто використати 6 **if**ів, з урахуванням пріоритетів (Рисунок 1.8).

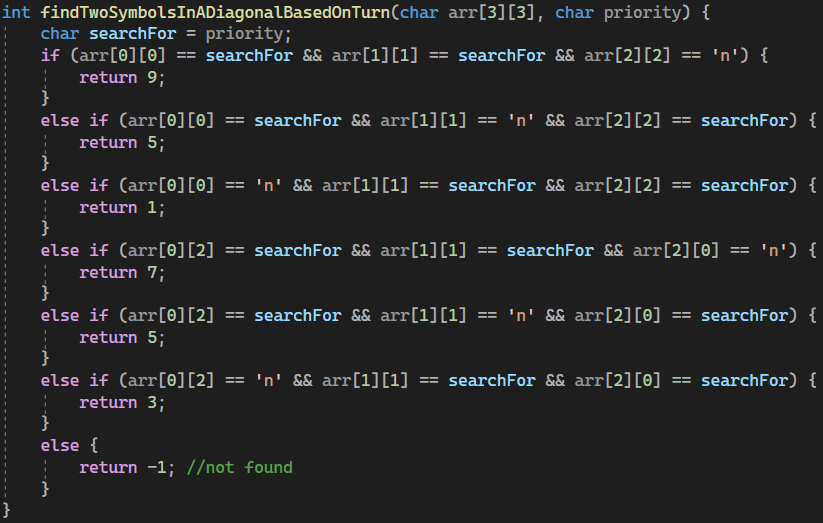


Рисунок 1.16 – пошук діагональних переможних позицій

**2 Основна частина:**

* опис виконаних робіт (завдань) навчальної практики;

Мною були розроблені наступні функції:

void displayArrViaPixels(char arr[][3], int size) – виведення масиву по пікселям

void fillCheckArr(bool(&checkArr)[9], int input) – заповнити checkArr

void checkIfElementExists(bool checkArr[9], int& input) – перевірити наявність елементів в масиві

void fillArrWithN(char(&arr)[3][3], int size) – очистити дошку

void displayArr(char arr[][3], int size) – вивести масив на екран

int checkWin(char arr[][3], int turn, int size) – перевірити перемогу

// 0 - game is still on

// 1 - x wins

// 2 - o wins

// 3 - draw

// turn 1 means turn o

// turn 0 means turn x)

void alertResult(char arr[][3], int turn, int size) – сповістити про результати гри

void botEasy(int& input, bool(&checkArr)[9]) – бот easy

void CreateRowColumnPatternsBasedOnPriority(char(&patternArr)[3][3], int size, char priority) – заповнити шаблон переможних позицій даним в залежності від пріорітету

int inputBasedOnRowColumnPattern(int pattern, int size, char arr[3][3]) – input при знаходженні переможної позиції

int findTwoSymbolsInARowBasedOnTurn – знайти переможну позицію в рядку

int findTwoSymbolsInAColumnBasedOnTurn(char arr[3][3], int size, char priority) – знайти переможну позицію в стовбці

int findTwoSymbolsInADiagonalBasedOnTurn(char arr[3][3], char priority) – знайти переможну позицію в діагоналі

void FindTwoCrosses(char arr[3][3], int size, int& input, string action) – знайти переможну позицію

void botMedium(int& input, bool(&checkArr)[9], char arr[3][3], int size, string action) – бот medium

void botHardFirstMove(char arr[3][3], string action, int& input) – перший хід бота hard

void botHardSecondMove(char arr[3][3], string action, int& input) – другий хід бота hard

void botHardThirdMove(char arr[3][3], string action, int& input) – третій хід бота hard

void copyArr(char arr[3][3], char(&tempArr)[3][3], int size) – скопіювати масив

void inputO(int input, char(&arr)[3][3]) – ввід ‘o’ в arr

void inputX(int input, char(&arr)[3][3]) – ввід ‘x’ в arr

int checkCheckArr(bool checkArr[9]) – перевірити кількість зайнятих клітинок

void botHard(int& input, bool(&checkArr)[9], char arr[3][3], int size, string action, int turn) – бот hard

void inputTurn(int& turn, int& input, char(&arr)[3][3], int size, bool(&checkArr)[9], string action, string difficulty) – прийняти хід та перенаправити гравця до гри в залежності від його вибору

void playerMode(string action) – режим гри з людина проти людини

void botMode(string action) – режим гри людина проти бота

void inputAction() – обрати режим гри

int main() – функція main

Також мною були розроблені наступні регулярні вирази:

regex botXReg("\\s\*bo?t?\\s\*x\\s\*"); //bot x b x

regex botOReg("\\s\*bo?t?\\s\*o\\s\*"); //bot o b o

regex playerReg("\\s\*pl?a?y?e?r?\\s\*\\s\*"); // player p

regex botXBotOReg("\\s\*bo?t?\\s\*x\\s\*|\\s\*bo?t?\\s\*o\\s\*"); //both bot x b x AND bot o b o

regex botXBotORegPlayerReg("\\s\*bo?t?\\s\*x\\s\*|\\s\*bo?t?\\s\*o\\s\*|\\s\*pl?a?y?e?r?\\s\*\\s\*"); // bot x b x AND bot o b o AND player p

regex botXBotORegPlayerRegStop("\\s\*bo?t?\\s\*x\\s\*|\\s\*bo?t?\\s\*o\\s\*|\\s\*pl?a?y?e?r?\\s\*\\s\*|stop"); // bot x b x AND bot o b o AND player p

* опис технічних та програмних засобів для їх виконання;

До програмних засобів можна віднести:

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Notepad++ — текстовий редактор, призначений для програмістів і тих, кого не влаштовує скромна функціональність Блокнота, що входить до складу Windows. Notepad++ базується на компоненті Scintilla (потужному компоненті для редагування), написаному на C++ з використанням тільки Windows API і STL, що забезпечує максимальну швидкість роботи при мінімальному розмірі програми. Інтерфейс Notepad++ є багатомовним (українська мова присутня). Серед особливостей програми — підсвічування синтаксису та підтримка великої кількості мов програмування (C, C++, Java, XML, HTML, PHP,JavaScript, ASCII, Visual Basic&nbs/VBScript, SQL, Ruby, CSS, Pascal, Perl і Python), багатомовна підтримка, робота з декількома документами.

* спеціальні програмні засоби, якщо такі розроблялися, для виконання робіт навчальної практики.

Не розроблялися.

**РЕЗУЛЬТАТИ ВИКОНАННЯ ЗАВДАНЬ ПРАКТИКИ**

**Лістинг коду:**

//#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <string>

#include <regex>

//#include <windows.h>

using namespace std;

regex botXReg("\\s\*bo?t?\\s\*x\\s\*"); //bot x b x

regex botOReg("\\s\*bo?t?\\s\*o\\s\*"); //bot o b o

regex playerReg("\\s\*pl?a?y?e?r?\\s\*\\s\*"); // player p

regex botXBotOReg("\\s\*bo?t?\\s\*x\\s\*|\\s\*bo?t?\\s\*o\\s\*"); //both bot x b x AND bot o b o

regex botXBotORegPlayerReg("\\s\*bo?t?\\s\*x\\s\*|\\s\*bo?t?\\s\*o\\s\*|\\s\*pl?a?y?e?r?\\s\*\\s\*"); // bot x b x AND bot o b o AND player p

void displayArrViaPixels(char arr[][3], int size) {

const int SIZE = 3;

constexpr int CROSS\_AND\_CIRCLE\_SIZE = 7;

const string cross[CROSS\_AND\_CIRCLE\_SIZE] = { " ■"," x x ■"," x x ■"," x ■"," x x ■"," x x ■"," ■" };

const string circle[CROSS\_AND\_CIRCLE\_SIZE] = { " ■", " ooo ■"," o o ■"," o o ■"," o o ■"," ooo ■", " ■" };

const string space = " ■";

string crossCircle[CROSS\_AND\_CIRCLE\_SIZE][SIZE];

// cross[0] circle[0] space

// cross[1] circle[1] space

// cross[2] circle[2] space

// cross[3] circle[3] space

// cross[4] circle[4] space

// cross[5] circle[5] space

// cross[6] circle[6] space

cout << "\n■■■■■■■■■■■■■■■■■■■■■■■■\n";

for (int k = 0; k < size; k++) {

for (int i = 0; i < size; i++) {

if (arr[k][i] == 'x') {

for (int j = 0; j < CROSS\_AND\_CIRCLE\_SIZE; j++) {

crossCircle[j][i] = cross[j];

}

}

if (arr[k][i] == 'o') {

for (int j = 0; j < CROSS\_AND\_CIRCLE\_SIZE; j++) {

crossCircle[j][i] = circle[j];

}

}

if (arr[k][i] == 'n') {

for (int j = 0; j < CROSS\_AND\_CIRCLE\_SIZE; j++) {

crossCircle[j][i] = space;

}

}

}

for (int i = 0; i < CROSS\_AND\_CIRCLE\_SIZE; i++) {

for (int j = 0; j < size; j++) {

cout << crossCircle[i][j];

}

cout << endl;

}

cout << "■■■■■■■■■■■■■■■■■■■■■■■■\n";

}

cout << '\n';

}

void fillCheckArr(bool(&checkArr)[9], int input) {

checkArr[input - 1] = true;

}

void checkIfElementExists(bool checkArr[9], int& input) {

while (checkArr[input - 1] == true) {

cout << "Wrong input. Please enter new input\n";

cin >> input;

}

}

//fill n

void fillArrWithN(char(&arr)[3][3], int size) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

arr[i][j] = 'n';

}

}

}

// Display array

void displayArr(char arr[][3], int size) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

cout << arr[i][j] << setw(3);

}

cout << '\n';

}

}

int checkWin(char arr[][3], int turn, int size) {

// 0 - game is still on

// 1 - x wins

// 2 - o wins

// 3 - draw

// turn 1 means turn o

// turn 0 means turn x

int countForOHorizontal = 0;

int countForOVertical = 0;

if (turn == 1) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (arr[j][i] == 'o') {

countForOVertical++;

}

}

if (countForOVertical == 3) {

return 2;

}

countForOVertical = 0;

}

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (arr[i][j] == 'o') {

countForOHorizontal++;

}

}

if (countForOHorizontal == 3) {

return 2;

}

countForOHorizontal = 0;

}

if (arr[1][1] == 'o') {

if (arr[0][0] == 'o' && arr[2][2] == 'o') {

return 2;

}

if (arr[2][0] == 'o' && arr[0][2] == 'o') {

return 2;

}

}

}

int countForXHorizontal = 0;

int countForXVertical = 0;

if (turn == 0) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (arr[j][i] == 'x') {

countForXVertical++;

}

}

if (countForXVertical == 3) {

return 1;

}

countForXVertical = 0;

}

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (arr[i][j] == 'x') {

countForXHorizontal++;

}

}

if (countForXHorizontal == 3) {

return 1;

}

countForXHorizontal = 0;

}

if (arr[1][1] == 'x') {

if (arr[0][0] == 'x' && arr[2][2] == 'x') {

return 1;

}

if (arr[2][0] == 'x' && arr[0][2] == 'x') {

return 1;

}

}

}

bool flag = true; //assume that the array is fully filled

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if ((arr[i][j] != 'x') && (arr[i][j] != 'o')) {

flag = false;

}

}

}

if (flag == true) {

return 3; //draw

}

return 0;

}

void alertResult(char arr[][3], int turn, int size) {

if (checkWin(arr, turn, size) == 1) {

cout << "X WON!\n";

}

if (checkWin(arr, turn, size) == 2) {

cout << "O WON!\n";

}

if (checkWin(arr, turn, size) == 3) {

cout << "DRAW!\n";

}

}

//void findTwoCrossesInARow(int arr[3][3], int size) {

//

//}

void botEasy(int& input, bool(&checkArr)[9]) {

cout << "easy\n";

srand(time(nullptr));

input = rand() % 9 + 1;

while (checkArr[input - 1] == true) {

input = rand() % 9 + 1;

}

// fillCheckArr(checkArr, input);

}

void CreateRowColumnPatternsBasedOnPriority(char(&patternArr)[3][3], int size, char priority) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (patternArr[i][j] != 'n') {

patternArr[i][j] = priority;

}

}

}

}

int inputBasedOnRowColumnPattern(int pattern, int size, char arr[3][3]) noexcept {

char target = 't';

for (int i = 0; i < size; i++) {

if (arr[pattern][i] == 'n') {

arr[pattern][i] = target;

}

}

int count = 0;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

count++;

if (arr[i][j] == target) {

return count; //input

}

}

}

cout << "SOMETHING WENT WRONG";

return -1;

}

int findTwoSymbolsInARowBasedOnTurn(char arr[3][3], int size, char priority) {

const int SIZE = 3;

char rowPatterns[SIZE][SIZE]{

{'n','?','?'},

{'?','n','?'},

{'?','?','n'},

};

CreateRowColumnPatternsBasedOnPriority(rowPatterns, size, priority);

int count = 0;

int findPattern = -1;

for (int k = 0; k < size; k++) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

//cout << arr[i][j] << ' ' << rowPatterns[k][j] << '\n';

if (arr[i][j] == rowPatterns[k][j]) {

count++;

}

}

findPattern++;

if (count == 3) {

return inputBasedOnRowColumnPattern(findPattern, size, arr);

}

count = 0;

}

findPattern = -1;

}

return -1; // not found

}

int inputBasedOnColumnParttern(int pattern, int size, char arr[3][3]) noexcept {

char target = 't';

for (int i = 0; i < size; i++) {

if (arr[i][pattern] == 'n') {

arr[i][pattern] = target;

}

}

int count = 0;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

count++;

if (arr[i][j] == target) {

return count; //input

}

}

}

cout << "SOMETHING WENT WRONG";

return -1;

}

int findTwoSymbolsInAColumnBasedOnTurn(char arr[3][3], int size, char priority) {

const int SIZE = 3;

char columnPatterns[SIZE][SIZE]{

{'n','?','?'},

{'?','n','?'},

{'?','?','n'},

};

CreateRowColumnPatternsBasedOnPriority(columnPatterns, size, priority);

int count = 0;

int findPattern = -1;

for (int k = 0; k < size; k++) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (arr[j][i] == columnPatterns[k][j]) {

count++;

}

}

findPattern++;

if (count == 3) {

return inputBasedOnColumnParttern(findPattern, size, arr);

}

count = 0;

}

findPattern = -1;

}

return -1; // not found

}

int findTwoSymbolsInADiagonalBasedOnTurn(char arr[3][3], char priority) {

char searchFor = priority;

if (arr[0][0] == searchFor && arr[1][1] == searchFor && arr[2][2] == 'n') {

return 9;

}

else if (arr[0][0] == searchFor && arr[1][1] == 'n' && arr[2][2] == searchFor) {

return 5;

}

else if (arr[0][0] == 'n' && arr[1][1] == searchFor && arr[2][2] == searchFor) {

return 1;

}

else if (arr[0][2] == searchFor && arr[1][1] == searchFor && arr[2][0] == 'n') {

return 7;

}

else if (arr[0][2] == searchFor && arr[1][1] == 'n' && arr[2][0] == searchFor) {

return 5;

}

else if (arr[0][2] == 'n' && arr[1][1] == searchFor && arr[2][0] == searchFor) {

return 3;

}

else {

return -1; //not found

}

}

void FindTwoCrosses(char arr[3][3], int size, int& input, string action) {

char priority = '?';

if (regex\_match(action, botOReg)) {

priority = 'o';

}

else if (regex\_match(action, botXReg)) {

priority = 'x';

}

int tempInput1 = findTwoSymbolsInARowBasedOnTurn(arr, size, priority);

int tempInput2 = findTwoSymbolsInAColumnBasedOnTurn(arr, size, priority);

int tempInput3 = findTwoSymbolsInADiagonalBasedOnTurn(arr, priority);

if (tempInput1 != -1) {

input = tempInput1;

return;

}

else if (tempInput2 != -1) {

input = tempInput2;

return;

}

else if (tempInput3 != -1) {

input = tempInput3;

return;

}

if (regex\_match(action, botOReg)) {

priority = 'x';

}

else if (regex\_match(action, botXReg)) {

priority = 'o';

}

int tempInput4 = findTwoSymbolsInARowBasedOnTurn(arr, size, priority);

int tempInput5 = findTwoSymbolsInAColumnBasedOnTurn(arr, size, priority);

int tempInput6 = findTwoSymbolsInADiagonalBasedOnTurn(arr, priority);

if (tempInput4 != -1) {

input = tempInput4;

return;

}

else if (tempInput5 != -1) {

input = tempInput5;

return;

}

else if (tempInput6 != -1) {

input = tempInput6;

return;

}

else {

input = -1; // call botEasy or smth

}

}

void botMedium(int& input, bool(&checkArr)[9], char arr[3][3], int size, string action) {

FindTwoCrosses(arr, size, input, action);

if (input == -1) { //if it did not find two symbols

botEasy(input, checkArr);

}

}

void botHardFirstMove(char arr[3][3], string action, int& input) {

if (regex\_match(action, botXReg)) {

input = 7; //best first moves are the corner moves. let's choose 7

}

else if (regex\_match(action, botOReg)) {

if (arr[1][1] == 'n') {

input = 5;

}

else {

input = 7; //why not

}

}

}

void botHardSecondMove(char arr[3][3], string action, int& input) {

if (regex\_match(action, botXReg)) {

if (arr[1][1] == 'o') {

input = 3; //why not

}

if (arr[0][0] == 'o' || arr[0][1] == 'o' || arr[1][0] == 'o') {

input = 9;

}

if (arr[2][1] == 'o' || arr[2][2] == 'o' || arr[1][2] == 'o') {

input = 1;

}

if (arr[0][2] == 'o') {

input = 9;

}

}

if (regex\_match(action, botOReg)) {

//general solutions

if (arr[0][0] == 'n') {

input = 1;

}

if (arr[0][1] == 'n') {

input = 1;

}

if (arr[0][2] == 'n') {

input = 1;

}

//////////////////////

if (arr[0][2] == 'x') {

input = 6;

}

if (arr[1][2] == 'x') {

input = 9;

}

if (arr[0][0] == 'x' && arr[2][2] == 'x' && arr[1][1] == 'o') {

input = 6;

}

if (arr[2][0] == 'x' && arr[0][2] == 'x' && arr[1][1] == 'o') {

input = 6;

}

}

}

void botHardThirdMove(char arr[3][3], string action, int& input) {

if (action == "bot x" || action == "b x") {

if (arr[1][1] == 'o') {

//do nothing (mediumBot will handle the situation)

}

if (arr[0][0] == 'x' && arr[2][0] == 'x') {

input = 3;

}

if (arr[0][2] == 'x' && arr[2][1]) {

//do nothing (mediumBot will handle the situation) since

// o n x

// n o x

// x n n

}

if (arr[0][2] == 'x' && arr[2][2] == 'x' && arr[0][1] == 'n') {

input = 3;

}

if (arr[0][2] == 'x' && arr[2][2] == 'x' && arr[0][1] == 'o') {

input = 1;

}

if (arr[2][0] == 'x' && arr[2][1] == 'o' && arr[2][2] == 'x') {

input = 1;

}

if (arr[0][0] == 'o' && arr[2][0] == 'x' && arr[2][1] == 'o' && arr[2][2] == 'x') {

input = 3;

}

if (arr[1][0] == 'o' && arr[2][0] == 'x' && arr[2][1] == 'o' && arr[2][2] == 'x') {

input = 3;

}

}

}

void copyArr(char arr[3][3], char(&tempArr)[3][3], int size) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

tempArr[i][j] = arr[i][j];

}

}

}

void inputO(int input, char(&arr)[3][3]) {

if (input == 1) {

arr[0][0] = 'o';

}

if (input == 2) {

arr[0][1] = 'o';

}

if (input == 3) {

arr[0][2] = 'o';

}

if (input == 4) {

arr[1][0] = 'o';

}

if (input == 5) {

arr[1][1] = 'o';

}

if (input == 6) {

arr[1][2] = 'o';

}

if (input == 7) {

arr[2][0] = 'o';

}

if (input == 8) {

arr[2][1] = 'o';

}

if (input == 9) {

arr[2][2] = 'o';

}

}

void inputX(int input, char(&arr)[3][3]) {

if (input == 1) {

arr[0][0] = 'x';

}

if (input == 2) {

arr[0][1] = 'x';

}

if (input == 3) {

arr[0][2] = 'x';

}

if (input == 4) {

arr[1][0] = 'x';

}

if (input == 5) {

arr[1][1] = 'x';

}

if (input == 6) {

arr[1][2] = 'x';

}

if (input == 7) {

arr[2][0] = 'x';

}

if (input == 8) {

arr[2][1] = 'x';

}

if (input == 9) {

arr[2][2] = 'x';

}

}

int checkCheckArr(bool checkArr[9]) {

int count = 0;

for (int i = 0; i < 9; i++) { //9 is the size of checkArr

if (checkArr[i] == true) {

count++;

}

}

return count;

}

void botHard(int& input, bool(&checkArr)[9], char arr[3][3], int size, string action, int turn) {

FindTwoCrosses(arr, size, input, action);

static int didFirstMoveHappen = 0;

static int didSecondMoveHappen = 0;

static int didThirdMoveHappen = 0;

if (checkCheckArr(checkArr) == 0) {

didFirstMoveHappen = 0;

didSecondMoveHappen = 0;

didThirdMoveHappen = 0;

}

if (regex\_match(action, botOReg)) {

if (checkCheckArr(checkArr) == 1) {

didFirstMoveHappen = 0;

didSecondMoveHappen = 0;

didThirdMoveHappen = 0;

}

}

if (input == -1) {

if (didFirstMoveHappen == 0) {

botHardFirstMove(arr, action, input);

didFirstMoveHappen++;

}

}

if (input == -1) {

if (didSecondMoveHappen == 0) {

botHardSecondMove(arr, action, input);

didSecondMoveHappen++;

}

}

if (input == -1) {

if (didThirdMoveHappen == 0) {

botHardThirdMove(arr, action, input);

}

}

//cout << "game: " << checkWin(arr, turn, size);

if (input == -1) {

botEasy(input, checkArr); //call the expert

}

}

void inputTurn(int& turn, int& input, char(&arr)[3][3], int size, bool(&checkArr)[9], string action, string difficulty) {

//перевірка x і o

if (turn == 1 && checkWin(arr, turn, size) == 0) {

checkIfElementExists(checkArr, input);

fillCheckArr(checkArr, input);

inputX(input, arr);

turn = 0;

//displayArr(arr, size);

displayArrViaPixels(arr, size);

//cout << checkWin(arr, turn, size);

if (checkWin(arr, turn, size) == 0) {

if (regex\_match(action, botOReg)) {

if (difficulty == "easy") {

botEasy(input, checkArr);

}

else if (difficulty == "medium") {

botMedium(input, checkArr, arr, size, action);

}

else if (difficulty == "hard") {

botHard(input, checkArr, arr, size, action, turn);

}

}

else {

cout << "Turn o: ";

cin >> input;

while (input < 1 || input > 9) {

cout << "Wrong. Enter input:\n";

cin >> input;

}

}

}

}

if (turn == 0 && checkWin(arr, turn, size) == 0) {

checkIfElementExists(checkArr, input);

fillCheckArr(checkArr, input);

inputO(input, arr);

turn = 1;

//displayArr(arr, size);

displayArrViaPixels(arr, size);

//cout << checkWin(arr, turn, size) << '\n';

if (checkWin(arr, turn, size) == 0) {

if (regex\_match(action, botXReg)) {

if (difficulty == "easy") {

botEasy(input, checkArr);

}

else if (difficulty == "medium") {

botMedium(input, checkArr, arr, size, action);

}

else if (difficulty == "hard") {

botHard(input, checkArr, arr, size, action, turn);

}

}

else {

cout << "Turn x: ";

cin >> input;

while (input < 1 || input > 9) {

cout << "Wrong. Enter input:\n";

cin >> input;

}

}

}

}

}

void playerMode(string action) {

int turn = 1; // 0 - o 1 - x

const int SIZE = 3;

char arr[SIZE][SIZE]{};

bool checkArr[] = { false,false,false,false,false,false,false,false,false };

fillArrWithN(arr, SIZE);

//displayArr(arr, size);

displayArrViaPixels(arr, SIZE);

int input;

cout << "Enter input:\n";

cout << "Turn x: ";

cin >> input;

while (input < 1 || input > 9) {

cout << "Wrong. Enter input:\n";

cin >> input;

}

string difficulty = "";

while (checkWin(arr, turn, SIZE) == 0) {

inputTurn(turn, input, arr, SIZE, checkArr, action, difficulty);

alertResult(arr, turn, SIZE);

}

}

void botMode(string action) {

int turn = 1; // 0 - o 1 - x

const int SIZE = 3;

char arr[SIZE][SIZE]{};

bool checkArr[] = { false,false,false,false,false,false,false,false,false };

fillArrWithN(arr, SIZE);

//displayArr(arr, size);

string difficulty;

cout << "Enter the difficulty. Available choices: easy, medium, hard\n";

cin >> difficulty;

while (difficulty != "easy" && difficulty != "medium" && difficulty != "hard") {

cout << "Wrong difficulty. Enter the difficulty. Available choices: easy, medium, hard\n";

cin >> difficulty;

}

displayArrViaPixels(arr, SIZE);

int input = -1;

if (regex\_match(action, botOReg)) {

cout << "Enter input:\n";

cout << "Turn x: ";

cin >> input;

}

while ((input < 1 || input > 9)) {

if (regex\_match(action, botXReg)) {

break;

}

cout << "Wrong. Enter input:\n";

cin >> input;

}

if (regex\_match(action, botOReg)) {

checkIfElementExists(checkArr, input);

fillCheckArr(checkArr, input);

inputX(input, arr);

turn = 0;

}

bool flag = 0;

if (flag == 0) {

flag = 1;

if (difficulty == "easy") {

botEasy(input, checkArr);

}

else if (difficulty == "medium") {

botMedium(input, checkArr, arr, SIZE, action);

}

else if (difficulty == "hard") {

botHard(input, checkArr, arr, SIZE, action, turn);

}

if (regex\_match(action, botXReg)) {

inputO(input, arr);

//turn = 0;

}

}

while (checkWin(arr, turn, SIZE) == 0) {

inputTurn(turn, input, arr, SIZE, checkArr, action, difficulty);

alertResult(arr, turn, SIZE);

}

}

void inputAction() {

string action = "something";

while (action != "stop") {

cout << "Would you like to play with a bot or with a player? If you want to play with a bot, type in 'bot' or 'b' with a corresponding symbol 'x' or 'o' e.g. 'bot x', if you want to play with a player, type in 'player' or 'p'. If you want to stop playing the game, type in 'stop'. \n";

getline(cin, action);

//cout <<"action: "<< action << '\n';

while (!regex\_match(action, botXBotORegPlayerRegStop)) {

cout << "Wrong action. If you want to play with a bot, type in 'bot' or 'b' with a corresponding symbol 'x' or 'o' e.g. 'bot x'. if you want to play with a player, type in 'player' or 'p'. If you want to stop playing the game, type in 'stop'. \n";

getline(cin, action);

}

if (regex\_match(action, playerReg)) {

playerMode(action);

}

if (regex\_match(action, botXBotOReg)) {

botMode(action);

}

if (action == "stop") {

return;

}

cin.ignore();

}

}

int main()

{

inputAction();

}

Приклад роботи програми:

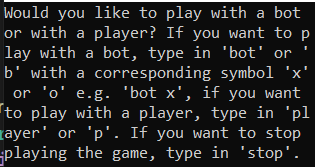


Рисунок 3.1 – інструкція користування програмою

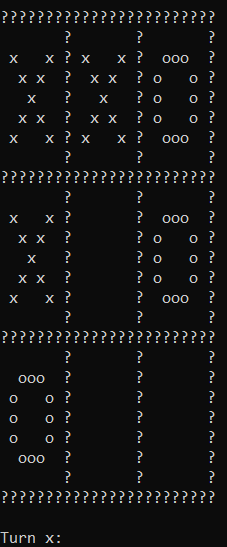


Рисунок 3.2 – хід гри людини проти людини

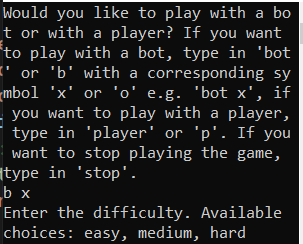


Рисунок 3.2 – обрання складності бота

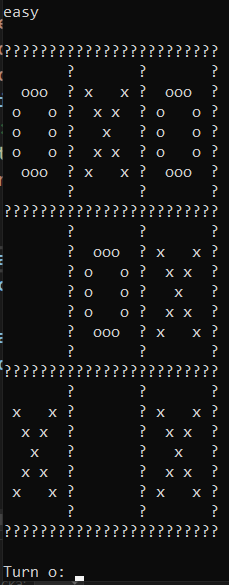


Рисунок 3.3 – гра проти бота easy

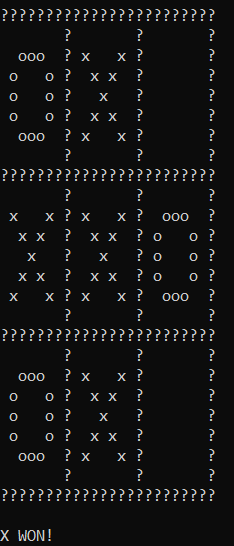


Рисунок 3.3 – гра проти бота medium

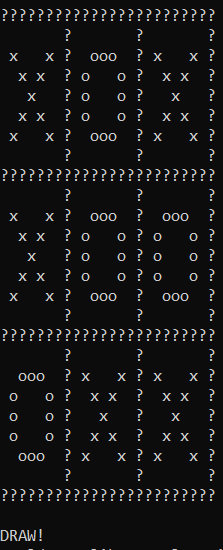


Рисунок 3.4 – гра проти бота hard

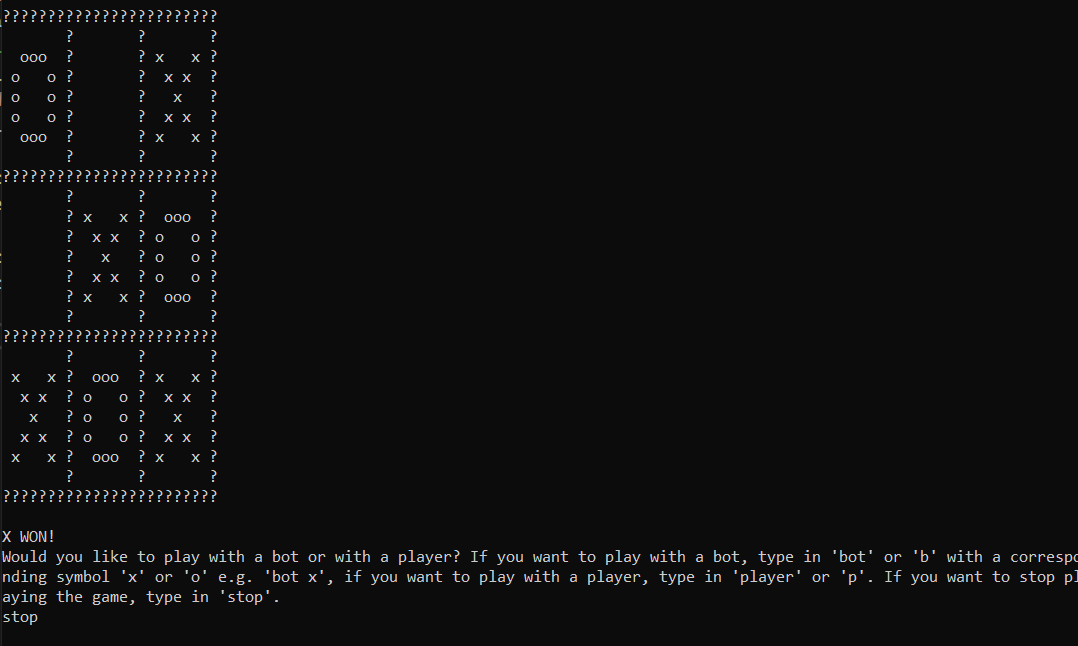


Рисунок 3.5 – кінець гри

# ВИСНОВКИ

В ході проходження практики були здобутті знання та досвід при роботі з функціями, вказівниками, регулярними виразами, conditional branching, статичними змінними та псевдографікою. Також було отримано досвід роботи з Visual Studio. Ознайомлено з алгоритмами перемоги в Tic-Tac-Toe за різні сторони гри. Отримано знання стосовно роботи з різноманітними алгоритмами. Здобутті уміння роботи з великими масивами. Отримано практику оформлення звітів. Здобуто досвід роботи з Notepad++. Отримано досвід роботи з ітерацією через масиви. Здобуті знання роботи з класом string та бібліотекою iomanip. Отримано досвід роботи з великими проектами.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1) <https://www.youtube.com/shorts/ZBZcnImNmhk>

2) <https://www.youtube.com/shorts/pzxNbqo6M3k>

3) <https://www.youtube.com/shorts/FF9V2TXEO5A>