**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КУРСОВА РОБОТА

на тему:

«Реалізація комп'ютерної гри «Сапер» з використанням технології Windows Forms»

з предмету об’єктно орієнтовне програмування С#

студента 2 курсу групи ПД-24

кафедри інженерії програмного забезпечення

Гаврилюка Валерія Олександровича

Викладач

старший викладач кафедри, аспірант

інженерії програмного забезпечення,

Гребенюк Віктор Вікторович

оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

-Київ 2020-

Зміст

[ВСТУП 3](#_Toc41015683)

[Розділ 1. Історія розвитку C# 4](#_Toc41015684)

[1.1. Поява С# 4](#_Toc41015685)

[1.2. Версії мови C# 6](#_Toc41015686)

[Розділ 2. Особливості мови С# 9](#_Toc41015687)

[2.1. Типи даних 9](#_Toc41015688)

[2.2. Методи та функції 10](#_Toc41015689)

[2.3. Властивості 10](#_Toc41015690)

[Розділ 3. Загальна Інформація 11](#_Toc41015691)

[Розділ 4. Опис структури для користувача інтерфейсу 12](#_Toc41015692)

[4.1. Головне меню гри 12](#_Toc41015693)

[4.2. Процес гри 14](#_Toc41015694)

[Розділ 5. Реалізація алгоритму гри «Сапер» 16](#_Toc41015695)

[5.1 Реалізація алгориту створення поля та заповнення його мінами 16](#_Toc41015696)

[5.2 Реалізація неможливості програти під час першого ходу 17](#_Toc41015697)

[5.3 Реалізація алгоритму програшу 18](#_Toc41015698)

[5.4 Реалізація алгоритму відкриття пустих комірок 19](#_Toc41015699)

[5.5 Реалізація алгоритму перевірки перемоги в грі 21](#_Toc41015700)

[5.6 Реалізація алгоритму вибору рівня складності 22](#_Toc41015701)

[Висновок 23](#_Toc41015702)

[Список використаної літератури 24](#_Toc41015703)

[Додаток 1 25](#_Toc41015704)

# ВСТУП

В представленій курсовій роботі представлено процес створення комп'ютерної гри «Сапер» з використанням мови C# та технології Windows Forms.

Метою даної курсової роботи є поглиблення знань в мові програмування C# та створення комп'ютерної гри «Сапер» з використанням технології Windows Forms

Задачі даної курсової роботи:

1) Створити комп'ютерну гру з використанням технології Windows Forms.

2) Описати вихідний код додатку.

# Розділ 1. Історія розвитку C#

## Поява С#

C# (вимовляється Сі шарп)-об'єктно-орієнтована мова програмування. Розроблений в 1998-2001 роках групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберга і Скотта Вільтаумота як мова розробки додатків для платформи Microsoft.NET Framework. Згодом був стандартизований як ECMA - 334 та ISO/IEC 23270.

C# відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, винятки, коментарі у форматі XML.

Перейнявши багато від своїх попередників-мов C++, Delphi, Модула, Smalltalk і, особливо, Java — С#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C# на відміну від C++ не підтримує множинне спадкування класів (тим часом допускається множинне спадкування інтерфейсів).

C# є дуже близьким родичем мови програмування Java. Мова Java була створена компанією Sun Microsystems, коли глобальний розвиток інтернету поставив завдання розосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу покажчиків без контролю виходу за межі). Для роззосереджених обчислень була створена концепція віртуальної машини і машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програми і апаратними інструкціями комп'ютера або іншого інтелектуального пристрою.

Java набула чималої популярності, і була ліцензована також і компанією Microsoft. Але з плином часу Sun почала звинувачувати Microsoft, що і при створенні свого клону Java робить її сумісною виключно з платформою Windows, чим суперечить самій концепції машинно-незалежного середовища виконання і порушує ліцензійну угоду. Microsoft відмовилася піти назустріч вимогам Sun, і тому з'ясування відносин набуло статусу судового процесу. Суд визнав позицію Sun справедливою, і зобов'язав Microsoft відмовитися від позаліцензійного використання Java.

У цій ситуації в Microsoft вирішили, користуючись своєю вагою на ринку, створити свій власний аналог Java — мова, в якому корпорація стане повновладним господарем. Ця новостворена мова отримала назву C#. Вона успадкувала від Java концепції віртуальної машини( середовище. Net), байт-коду (WM) і більшої безпеки вихідного коду програм, плюс врахувала досвід використання Програм на Java.

Нововведенням C# стала можливість більш легкої взаємодії, в порівнянні з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проектів. Якщо Програми на різних мовах виконуються на платформі. NET,. NET бере на себе клопоти по сумісності програм (тобто типів даних, по кінцевому рахунку).

Символ # у назві мови можна інтерпретувати і як дві пари плюсів ++, що натякають на новий крок у розвитку мови порівняно з C++ (подібно до кроку від C до C++), і як музичний символ дієз, разом з буквою C, що становить в англійській мові назву ноти до-дієз. Останнє й дало назву мові. Попри те, що символ # (октоторп) насправді є символом для позначення номера на більшості клавіатур і відрізняється від символу дієз ♯ (Unicode U+266F), Microsoft, як автор мови, неодноразово зверталася до своїх клієнтів з проханням прийняти таку стилізацію.

## Версії мови C#

***Версія 1.0***

Проект c# був розпочатий в грудні 1998 і отримав кодову назву COOL (C-style Object Oriented Language). Версія 1.0 була анонсована разом з платформою .Net в червні 2000 року, тоді ж з'явилася і перша загальнодоступна бета-версія; C# 1.0 остаточно вийшов разом з Microsoft Visual Studio. Net в лютому 2002 року.

Перша версія C# нагадувала за своїми можливостями Java 1.4, кілька розширюючи їх: так, в C# малися властивості (виглядають в коді як поля об'єкта, але на ділі викликають при зверненні до них методи класу), індексатори (подібні властивостям, але беруть параметр як індекс масиву), події, делегати, цикли foreach, структури, що передаються за значенням, автоматичне перетворення вбудованих типів в об'єкти при необхідності (boxing), атрибути, вбудовані засоби взаємодії з некерованим кодом (DLL, COM) і інше.

Крім того, в C# вирішено було перенести деякі можливості C++, відсутні в Java: беззнакові типи, перевантаження операторів (з деякими обмеженнями, на відміну від C++), передача параметрів в метод за посиланням, методи зі змінним числом параметрів, оператор goto (з обмеженнями). Також в C# залишили обмежену можливість роботи з покажчиками — в місцях коду, спеціально позначених словом unsafe і при вказівці спеціальної опції компілятору.

***Версія 2.0***

Проект специфікації C# 2.0 вперше був викладений Microsoft в жовтні 2003 року; у 2004 році виходили бета-версії (проект з кодовою назвою Whidbey), C# 2.0 остаточно вийшов 7 листопада 2005 року разом з Visual Studio 2005 і .NET 2.0.

Нові можливості у версії 2.0:

* Часткові типи (розділення реалізації класу більш ніж на один файл).
* Узагальнені, або параметризовані типи (generics, «дженерики»). На відміну від шаблонів C++, вони підтримують деякі додаткові можливості і працюють на рівні віртуальної машини. Разом з тим, параметрами узагальненого типу не можуть бути вирази.
* Нова форма ітератора, що дозволяє створювати співпрограми за допомогою ключового слова *yield*, подібно [Python](https://uk.wikipedia.org/wiki/Python" \o "Python) і [Рубі](https://uk.wikipedia.org/wiki/%D0%A0%D1%83%D0%B1%D1%96).
* Анонімні методи, що забезпечують функціональність замикання.
* Оператор ??: return obj1 ?? obj2; означає (у нотації C# 1.0) return obj1!=null ? obj1 : obj2;.
* Типи-значення, що обнуляються (nullable), (що позначаються знаком питання, наприклад, int? i = null;) є тими ж самими типами-значеннями, що можуть також приймати також значення null. Такі типи дозволяють поліпшити взаємодію з базами даних через мову SQL.

***Версія 3.0***

У червні 2004 року Андерс Гейлсберг вперше розповів на сайті Microsoft про плановані розширення мови C#3.0.. У вересні 2005 року був випущений проект специфікації C# 3.0 і бета-версія C# 3.0, яка встановлюється у вигляді доповнення до існуючих Visual Studio 2005 і.NET 2.0. Офіційно версія C # 3.0 побачила світ 19 листопада 2007 року в складі.NET Framework 3.5.

У C# 3.0 з'явилися такі радикальні доповнення і зміни:

* Ключові слова select, from, where, що дозволяють робити запити з SQL, XML, колекції тощо (запит, інтегрований в мову — LINQ, англ. Language Integrated Query);
* Ініціалізація об'єкта разом з його властивостями;
* Лямбда вирази;
* Автоматичне визначення типів локальних змінних;

C# 3.0 сумісний з C# 2.0 по генерованим WM-коду; поліпшення в мові — чисто синтаксичні і реалізуються на етапі компіляції. Наприклад, багато з інтегрованих запитів LINQ можна реалізувати в поточних версіях використовуючи безіменні делегати в поєднанні з предикативними методами над контейнерами на зразок List.FindAll і List.RemoveAll.

***Версія 4.0***

Випуск четвертої версії мови програмування C# не випадково збігся з випуском нової версії.NET Framework. Він мав на меті створення інфраструктури для реалізації мов динамічної типізації. Що істотно поліпшило підтримку динамічних API в C # , було бажаним для початку робіт над мовою TypeScript і було використано наявними реалізаціями, наприклад PowerShell

***Версія 5.0***

* Асинхронні методи
* Відомості про об'єкт, що робить виклик

# Розділ 2. Особливості мови С#

## 2.1. Типи даних

C# підтримує строго типізовані неявні оголошення змінних з ключовим словом var і неявно типізовані масиви з ключовим словом new [], за яким слідує ініціалізатор колекції. C# підтримує суворий тип даних Boolean, bool. Вирази, які приймають умови, такі як while та if, вимагають висловлювання, що реалізує оператор true або false. Хоча C++ також має тип Boolean, він може бути вільно перетворений в цілі числа та з них, а вирази, такі як if(a), вимагають тільки того, щоб a був конвертований в bool, що дозволяє бути a int-типу або вказівником. C# забороняє «ціле значення означає справжній або помилковий підхід» на тій підставі, що примус програмістів використовувати вирази, які повертають точно bool, можуть створювати деякі типи помилок програмування, наприклад if (a = b) (використання присвоювання = замість рівності == , які, хоча і не є помилкою на C або C ++, все одно будуть спіймані компілятором). C# безпечніший в порівнянні з C++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявних перетворень між булевими і цілими числами, а також між членами перерахування і цілими числами (крім літерала 0, який може бути неявно перетворений в будь-який нумерований тип). Будь-яке призначене для користувача перетворення повинно бути явно позначене як явне або неявне, на відміну від конструкторів копіювання C++ і операторів перетворення, які за умовчанням є неявними. C# має явну підтримку ковариації та контраваріантності в родових типах, на відміну від C++, яка має певний рівень підтримки контраваріантності просто через семантику типів,що повертаються, на віртуальні методи. Члени перерахування розміщуються в своєму власному обсязі. Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні та функції.

## 2.2. Методи та функції

Методи в мові програмування є членами класу в проекті, деякі методи мають підписи, а деякі не мають підпису. Методи можуть бути недійсними або можуть повертати щось на зразок рядка, цілого, подвійного, десяткового, float і bool. Якщо метод недійсний, це означає, що метод не повертає жодного типу даних. Подібно C++, і на відміну від Java, програмісти на C # повинні використовувати ключове слово virtual, щоб дозволити перевизначати методи підкласами[1]. Методи розширення в C# дозволяють програмістам використовувати статичні методи, як якщо б вони були методами з таблиці методів класу, дозволяючи програмістам додавати методи до об'єкта, який, на їхню думку, повинен існувати на цьому об'єкті і його похідних. Динамічний тип dynamic допускає прив'язку методу під час виконання, що дозволяє використовувати JavaScript-подібні виклики методів і склад часу виконання. У C# є підтримка строго типізованих покажчиків функцій через delegate ключового слова. Подібно псевдо-C ++ - signal і slot фрейма Ct, C# має семантику, спеціально пов'язану з подіями стилю публікації-підписки, хоча C# використовує делегати для цього. C # пропонує Java-подібні синхронізовані synchronized виклики методів через атрибут [MethodImpl (MethodImplOptions.Synchronized)] і підтримує взаємовиключні блокування за допомогою блокування ключових слів.

## 2.3. Властивості

C# надає властивості як синтаксичного цукру для загального шаблону, в якому пара методів, accessor (getter) і mutator (setter) інкапсулює операції по одному атрибуту класу. Не потрібно писати надлишкові сигнатури методів для реалізацій геттера / сетера і до цієї властивості можна отримати доступ, використовуючи синтаксис атрибутів, а не більш докладні виклики методів.

# Розділ 3. Загальна Інформація

Для створення інтерфейсу користувача буде використано інтерфейс програмування додатків Windows Forms.

Windows Forms - інтерфейс програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код - класи, що реалізують API для Windows Forms, що не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні ПЗ на C #, С ++, так і на VB.Net, J # і ін.

Як і Abstract Window Toolkit (AWT) (схожий API для мови Java), бібліотека Windows Forms була розроблена як частина .NET Framework для спрощення розробки компонентів графічного інтерфейсу користувача. Windows Forms побудована на основі застарілого Windows API.

Windows Forms надає можливість розробки кросплатформного графічного користувацького інтерфейсу. Але Windows Forms є лише обгорткою Windows API-компонентів.

# Розділ 4. **Опис структури для користувача інтерфейсу**

## 4.1. Головне меню гри

На рис 1.1 зображено головне меню гри «Сапер», в якому можна обрати рівень складності гри, від якого залежить розмір ігрового поля (рис 1.2-1.4)

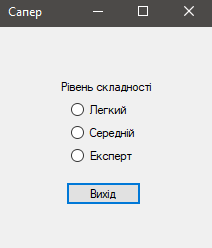
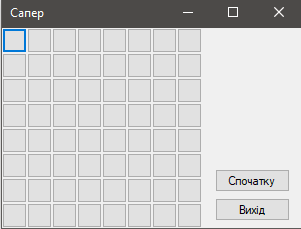


Рис. 1.1. Головне меню гри.



1.2 Легкий рівень складності (Поле 8х8)

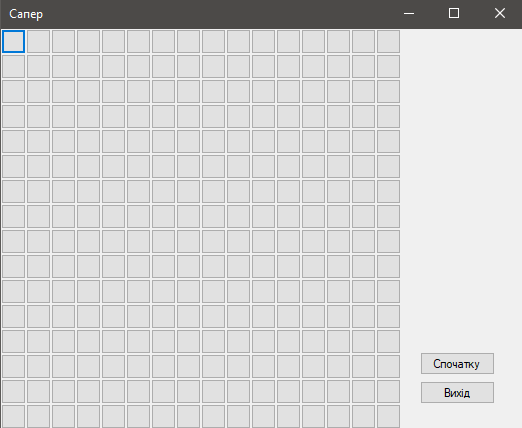


Рис 1.3 Середній рівень складності (Поле 16х16)



Рис 1.4 Рівень складності Експерт (Ігрове поле 30х16)

## 4.2. Процес гри

Гра починається з відкриття однієї комірки (Рис 2.1). Число в комірці показує, скільки приховано мін навколо даної комірки (Рис 2.2). Якщо ви відкрили комірку з міною, то гра програна (Рис 2.3). Що б помітити комірку в якій знаходиться бомба, потрібно відмітити її правою кнопкою миші (Рис 2.4).

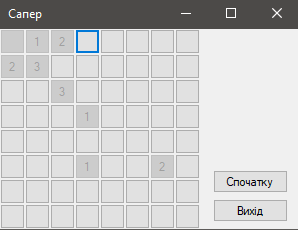


Рис 2.1 Початок гри

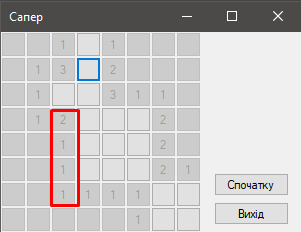


Рис 2.2 Кількість мін поруч з коміркою

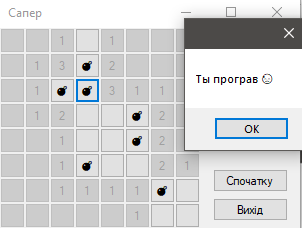


Рис 2.3 Програш гри

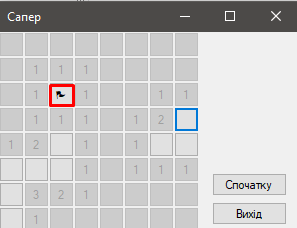


Рис 2.4 Мітка комірки з міною

# Розділ 5. Реалізація алгоритму гри «Сапер»

## 5.1 Реалізація алгориту створення поля та заповнення його мінами

На рис 3.1 можна побачити реалізацію алгоритма створення поля та заповнення його мінами.

Random random = new Random(); - Використовується для заповнення комірок мінами;

Змінні height та width відповідають за кількість ігрових комірок

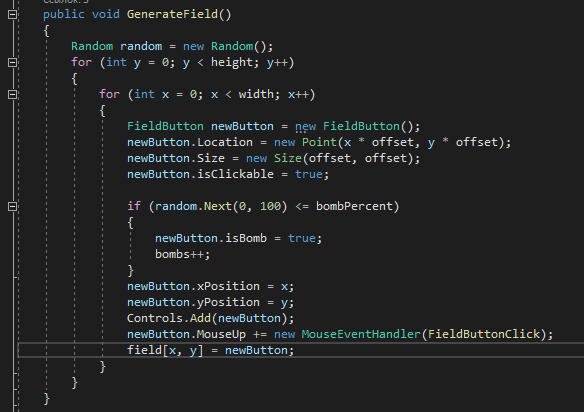


Рис 3.1 Алгоритм створення поля та заповнення його мінами

## 5.2 Реалізація неможливості програти під час першого ходу

На рис 3.2 можна побачити реалізацію алгориту неможливості програти під час першого ходу. Алгоритм полягає в тому, якщо перший клік буде здійснено на комірку під якою знаходиться міна, то вона видаляється і відкривається пуста комірка. Якщо клікнути на комірку з міною після першого ходу, то міна відкривається і гравець програє.

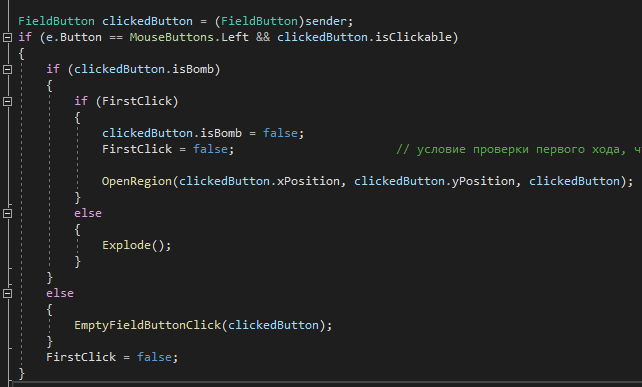


Рис 3.2 Алгоритм неможливості програти з першого ходу

## 5.3 Реалізація алгоритму програшу

На рис 3.3 можна побачити алгоритм програшу в грі. Алгоритм полягає в тому, що гравець відкриваючи комірку з бомбою і це не перший хід, то відкриваються всі комірки з бомбами і висвічується повідомлення на екрані «Ти програв 😓» і пропонується почати гру спочатку.

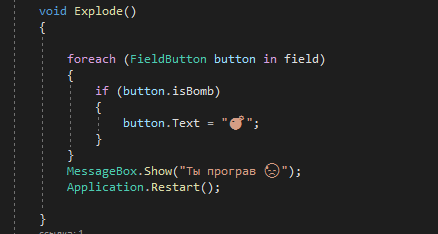


Рис 3.3 Алгоритм програшу гри

## 5.4 Реалізація алгоритму відкриття пустих комірок

На рис 3.4-3.5 можна побачити реалізацію алгоритму відкриття пустих комірок.

Алгоритм полягає в тому, що якщо клік буде здійснено на комірку без міни, то вона відкривається і гра продовжується, якщо в сусідніх комірках відсутні міни, то відкриваються комірки до тих пір, поки біля відкритих комірок не з’являться міни

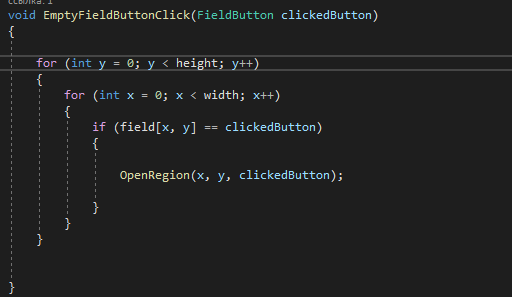


Рис 3.4 Алгоритм відкриття пустої комірки

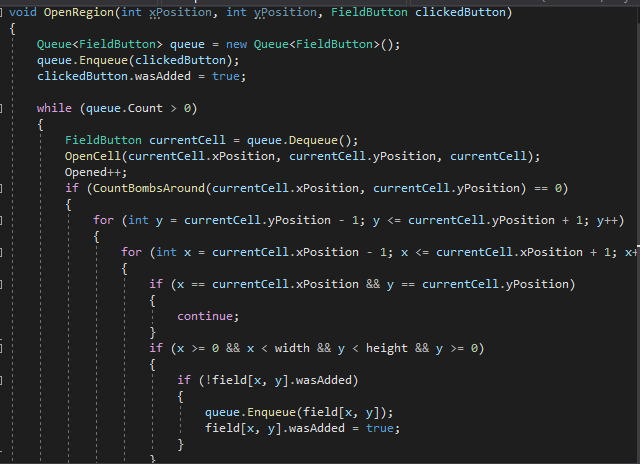
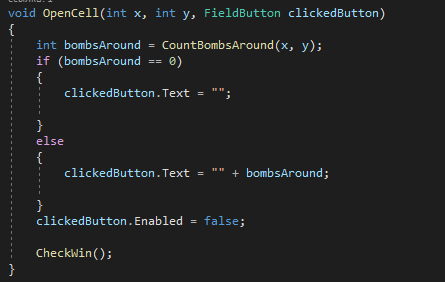


Рис 3.5 Алгоритм відкриття сусідніх порожніх комірок

На рис 3.6 можна побачити реалізацію відображення кількості мін які знаходяться по сусідству. Він був реалізований циклом, що підраховує кількість мін , що знаходиться в радіусі 3х3 навколо відкритої комірки і відображає їх кількість.



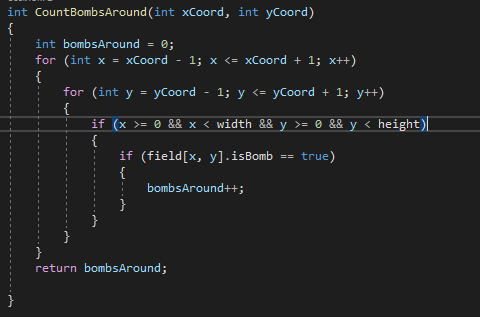


Рис 3.6 Алгоритм відображення кількості мін по сусідству біля відкритої комірки

## 5.5 Реалізація алгоритму перевірки перемоги в грі

Алгоритм був реалізований перевіркою кількості відкритих комірок, не враховуючи комірки з мінами. Якщо всі порожні комірки будуть відкриті, то на екрані відображається повідомлення «Перемога! 🎂»

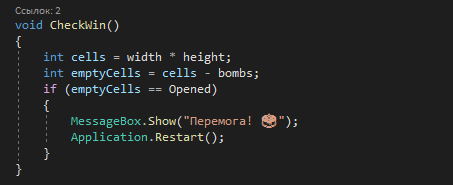


Рис 3.7 Алгоритм перевірки перемоги в грі

## 5.6 Реалізація алгоритму вибору рівня складності

Алгоритм був реалізований за допомогою radioButton, при виборі одного з них встановлюються параметри поля:

height – кількість рядків;

width – кількість стовпчиків;

bombPercent – відсоток появлення міни на полі

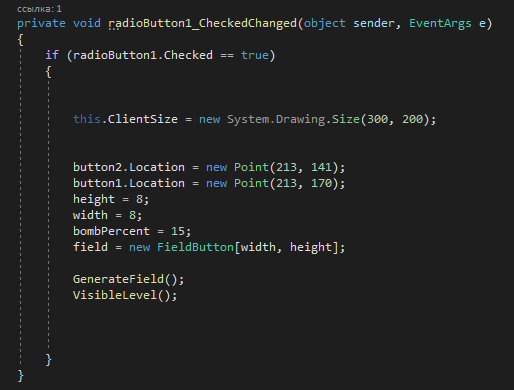


Рис 3.8 Алгоритм створення поля легкого рівня складності

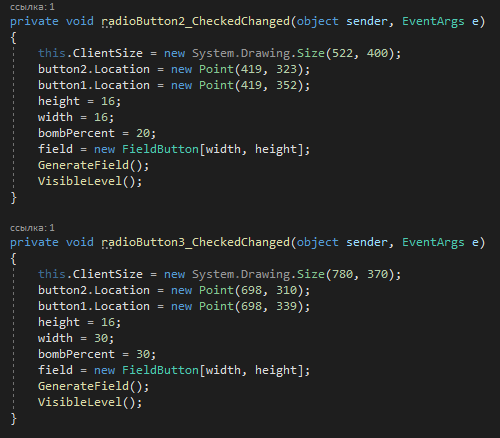


Рис 3. Алгоритм створення поля середньго та складного рівня складності

# Висновок

В ході виконання курсової роботи мовою програмування С# з використанням технології WindowsForms в середовищі VisualStudio 2019 була реалізована гра «Сапер». Також були поглибленні та закріплені знання в об’єктно-орієнтовному програмуванні. Данну гру, яка була реалізована під час виконання курсової роботи, можна використовувати для покращення логіки та веселого проводження вільного часу.

# Список використаної літератури

1. Полное руководство C# 4. 0 Герберт Шилдт «Вильямс». Москва. Санкт-Петербург. Киев. 2007 р. – 375 с.
2. Троелсен, Эндрю Язык программирования C# 6.0 и платформа .NET 4.6 / Эндрю Троелсен , Филипп Джепикс. - М.: Вильямс, 2016. - 134 c
3. Дэвис, Алекс Асинхронное программирование в C# 5.0 / Алекс Дэвис. - М.: ДМК Пресс, 2015. - 120 c.
4. Зиборов, В. Visual C# 2012 на примерах / В. Зиборов. - М.: БХВ-Петербург, 2015. - 335 c.
5. Гэри, Маклин Холл Адаптивный код на C#. Проектирование классов и интерфейсов, шаблоны и принципы SOLID / Гэри Маклин Холл. - М.: Вильямс, 2015. - 432 c.
6. Введение в Windows Forms [Електронний ресурс]: <https://metanit.com/sharp/windowsforms/1.1.php>
7. Програма на С# .Net [Електронний ресурс]: <https://itproger.com/course/csharp-app>
8. Руководство по программированию в WindowsForms [Електронний ресурс]: https://metanit.com/sharp/windowsforms/

# Додаток 1

Вихідний код:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace Saper

{

public partial class Form1 : Form

{

int width;

int height;

int offset = 25;

int bombPercent;

bool FirstClick = true;

FieldButton[,] field;

int Opened = 0;

int bombs = 0;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

public void GenerateField()

{

Random random = new Random();

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

FieldButton newButton = new FieldButton();

newButton.Location = new Point(x \* offset, y \* offset);

newButton.Size = new Size(offset, offset);

newButton.isClickable = true;

if (random.Next(0, 100) <= bombPercent)

{

newButton.isBomb = true;

bombs++;

}

newButton.xPosition = x;

newButton.yPosition = y;

Controls.Add(newButton);

newButton.MouseUp += new MouseEventHandler(FieldButtonClick);

field[x, y] = newButton;

}

}

}

void FieldButtonClick(object sender, MouseEventArgs e)

{

FieldButton clickedButton = (FieldButton)sender;

if (e.Button == MouseButtons.Left && clickedButton.isClickable)

{

if (clickedButton.isBomb)

{

if (FirstClick)

{

clickedButton.isBomb = false;

FirstClick = false; // условие проверки первого хода, чтобы первое нажатие всегда было на пустую клетку

OpenRegion(clickedButton.xPosition, clickedButton.yPosition, clickedButton);

}

else

{

Explode();

}

}

else

{

EmptyFieldButtonClick(clickedButton);

}

FirstClick = false;

}

if (e.Button == MouseButtons.Right)

{

clickedButton.isClickable = !clickedButton.isClickable; // Метка бомбы правой кнопкой мыши

if (!clickedButton.isClickable)

{

clickedButton.Text = "🏲";

}

else

{

clickedButton.Text = "";

}

}

CheckWin();

}

void Explode()

{

foreach (FieldButton button in field)

{

if (button.isBomb)

{

button.Text = "💣";

}

}

MessageBox.Show("Ты програв 😓");

Application.Restart();

}

void EmptyFieldButtonClick(FieldButton clickedButton)

{

for (int y = 0; y < height; y++)

{

for (int x = 0; x < width; x++)

{

if (field[x, y] == clickedButton)

{

OpenRegion(x, y, clickedButton);

}

}

}

}

void OpenRegion(int xPosition, int yPosition, FieldButton clickedButton)

{

Queue<FieldButton> queue = new Queue<FieldButton>();

queue.Enqueue(clickedButton);

clickedButton.wasAdded = true;

while (queue.Count > 0)

{

FieldButton currentCell = queue.Dequeue();

OpenCell(currentCell.xPosition, currentCell.yPosition, currentCell);

Opened++;

if (CountBombsAround(currentCell.xPosition, currentCell.yPosition) == 0)

{

for (int y = currentCell.yPosition - 1; y <= currentCell.yPosition + 1; y++)

{

for (int x = currentCell.xPosition - 1; x <= currentCell.xPosition + 1; x++) // функция автоматического раскрытия всех соседних пустых клеток

{

if (x == currentCell.xPosition && y == currentCell.yPosition)

{

continue;

}

if (x >= 0 && x < width && y < height && y >= 0)

{

if (!field[x, y].wasAdded)

{

queue.Enqueue(field[x, y]);

field[x, y].wasAdded = true;

}

}

}

}

}

}

}

void OpenCell(int x, int y, FieldButton clickedButton)

{

int bombsAround = CountBombsAround(x, y);

if (bombsAround == 0)

{

clickedButton.Text = "";

}

else

{

clickedButton.Text = "" + bombsAround;

}

clickedButton.Enabled = false;

CheckWin();

}

int CountBombsAround(int xCoord, int yCoord)

{

int bombsAround = 0;

for (int x = xCoord - 1; x <= xCoord + 1; x++)

{

for (int y = yCoord - 1; y <= yCoord + 1; y++)

{

if (x >= 0 && x < width && y >= 0 && y < height)

{

if (field[x, y].isBomb == true)

{

bombsAround++;

}

}

}

}

return bombsAround;

}

void CheckWin()

{

int cells = width \* height;

int emptyCells = cells - bombs;

if (emptyCells == Opened)

{

MessageBox.Show("Перемога! 🎂");

Application.Restart();

}

}

void VisibleLevel()

{

radioButton1.Visible = false;

radioButton2.Visible = false;

radioButton3.Visible = false;

label1.Visible = false;

}

private void button1\_Click(object sender, EventArgs e)

{

MessageBoxButtons msb = MessageBoxButtons.YesNo;

String message = "Ви дійсно бажаєте вийти?";

String caption = "Вихід";

if (MessageBox.Show(message, caption, msb) == DialogResult.Yes)

this.Close();

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Restart();

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton1.Checked == true)

{

this.ClientSize = new System.Drawing.Size(300, 200);

button2.Location = new Point(213, 141);

button1.Location = new Point(213, 170);

height = 8;

width = 8;

bombPercent = 15;

field = new FieldButton[width, height];

GenerateField();

VisibleLevel();

}

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

this.ClientSize = new System.Drawing.Size(522, 400);

button2.Location = new Point(419, 323);

button1.Location = new Point(419, 352);

height = 16;

width = 16;

bombPercent = 20;

field = new FieldButton[width, height];

GenerateField();

VisibleLevel();

}

private void radioButton3\_CheckedChanged(object sender, EventArgs e)

{

this.ClientSize = new System.Drawing.Size(780, 370);

button2.Location = new Point(698, 310);

button1.Location = new Point(698, 339);

height = 16;

width = 30;

bombPercent = 30;

field = new FieldButton[width, height];

GenerateField();

VisibleLevel();

}

private void label1\_Click(object sender, EventArgs e)

{

}

public class FieldButton : Button

{

public bool isBomb;

public bool isClickable;

public bool wasAdded;

public int xPosition;

public int yPosition;

}

}

}