**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КУРСОВА РОБОТА

на тему:

«Автогенерація лабіринта»

з предмету «Об’єктно-орієнтовне програмування С#»

студента 2 курсу групи ПД-23

кафедри інженерії програмного забезпечення

Шевченко Ілля Сергійович

Викладач

аспірант, старший викладач кафедри

інженерії програмного забезпечення,

Гребенюк Віктор Вікторович

оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2023

# Вступ

## **Обґрунтування вибору теми та актуальність дослідження.**

Дослідження автогенерації лабіринтів на мові програмування C# може мати актуальність в різних контекстах і для різних цілей. Ось кілька причин, чому це дослідження може бути важливим:

Візуалізація та графічне представлення: Автогенерація лабіринтів може бути корисною в графічних додатках та ігровому розвитку. Генерація лабіринтів може створювати цікаві геймплейні ситуації та викликати взаємодію з гравцем.

Алгоритми пошуку та навігації: Автогенерація лабіринтів може послужити для вивчення та покращення алгоритмів пошуку шляху і навігації. Це має значення для робототехніки, штучного інтелекту та розумних агентів.

Тестування програм: Лабіринти можуть бути використані для створення тестових середовищ для програм, особливо тих, що мають складні алгоритми навігації або розташування об'єктів у просторі.

Педагогічні цілі: Дослідження автогенерації лабіринтів на мові програмування C# може бути корисним для використання в навчальних цілях, де студенти можуть вивчати алгоритми, графічне програмування та геймдевелопмент.

Враховуючи широкий спектр можливих застосувань, дослідження автогенерації лабіринтів на мові програмування C# може мати практичне значення та сприяти розвитку зазначених галузей.

## **Мета та завдання дослідження.**

Мета дослідження автогенерації лабіринтів полягає у вивченні та розробці алгоритмів, що дозволяють створювати лабіринти автоматично без прямого втручання людини. Автогенерація лабіринтів має потенціал застосування у багатьох областях, таких як графічне проектування, комп'ютерні ігри, штучний інтелект, дослідження алгоритмів пошуку шляху та інші.

Завдання дослідження автогенерації лабіринтів можуть включати:

Аналіз існуючих алгоритмів генерації лабіринтів: перегляд та вивчення попередніх робіт, алгоритмів та методів, що використовуються для створення лабіринтів.

Розробка нових алгоритмів: розроблення та впровадження нових алгоритмів, які забезпечують автоматичне створення лабіринтів. Це може включати в себе різні підходи, такі як алгоритми засновані на графах, генетичні алгоритми, клітинні автомати та інші.

Оцінка якості згенерованих лабіринтів: розроблення метрик та критеріїв для оцінки якості створених лабіринтів. Це може включати критерії, які оцінюють рівень складності лабіринту, його унікальність, кількість різних шляхів і т. д.

Практична реалізація: реалізація розроблених алгоритмів у вигляді програмного забезпечення або бібліотеки, що може бути використана для генерації лабіринтів у різних додатках або системах.

Експерименти та порівняння: проведення експериментів для порівняння різних алгоритмів та оцінки їхньої ефективності та продуктивності. Це дозволить визначити найкращі підходи до автогенерації лабіринтів і встановити рекомендації для їхнього використання.

Отже, дослідження автогенерації лабіринтів спрямоване на розвиток нових алгоритмів та методів для автоматичного створення лабіринтів, їх якісну оцінку і практичну реалізацію з метою використання в різних областях.

# Теоретична частина

## **Короткий огляд історії автогенерації лабіринтів.**

Автогенерація лабіринтів є досить давнім та популярним дослідженням в галузі комп'ютерних ігор та обчислювальної геометрії. Історія автогенерації лабіринтів сягає ранніх етапів розвитку комп'ютерної графіки та обчислювальних систем. Одним з ранніх прикладів автогенерації лабіринтів є гра "Рогатка" (Maze War), розроблена в 1973 році. Гра дозволяла гравцям переміщатися по тривимірному лабіринту, який автоматично генерувався на основі випадкових алгоритмів. Згодом, з'явилися інші ігри, такі як "Робот-король" (RobotWar) і "Дум" (Doom), які також використовували автоматичну генерацію лабіринтів. З розвитком графічних технологій і 3D-графіки, автогенерація лабіринтів стала ще більш популярною та складною.

Сучасні підходи до автогенерації лабіринтів використовують різні алгоритми, такі як алгоритми затоплення (flood fill), рекурсивні поділі лабіринту (recursive division), випадкове блукання (random walk), алгоритми пошуку шляху (pathfinding) та генетичні алгоритми. Ці алгоритми дозволяють створювати лабіринти з різними рівнями складності та унікальними структурами.

Дослідження в галузі автогенерації лабіринтів продовжуються і дослідники використовують їх у різних контекстах, таких як ігровий дизайн, штучний інтелект, комп'ютерна графіка та віртуальна реальність. Завдяки постійному розвитку комп'ютерних технологій та алгоритмів, автогенерація лабіринтів продовжує надихати нові дослідження та ігрові досвіди.

## **Опис алгоритму автогенерації алгоритму.**

Існує безліч алгоритмів для автогенерації лабіринтів, і кожен з них має свої особливості. Один з найпоширеніших алгоритмів для автогенерації лабіринтів - це алгоритм "Рекурсивне ділення" (Recursive Division). Давайте розглянемо його опис.

Крок 1: Початковою структурою є повний прямокутник, який представляє весь лабіринт.

Крок 2: Виберіть вертикальну або горизонтальну лінію, яка буде розділяти прямокутник навпіл. Ця лінія повинна пройти через випадкову позицію від одного краю до іншого.

Крок 3: Створіть прохід уздовж розміченої лінії. Це може бути просто відкритий прохід або вхід-вихід, залежно від ваших потреб.

Крок 4: Рекурсивно застосовуйте алгоритм до двох нових підпрямокутників, які утворилися внаслідок розділення. Тобто, викликайте алгоритм для лівої частини прямокутника і для правої частини (або верхньої і нижньої, якщо розділення було горизонтальним).

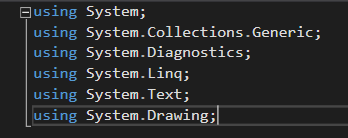
Крок 5: Продовжуйте рекурсивно застосовувати алгоритм до нових підпрямокутників, поки їх розмір не стане надто малим для поділу (наприклад, досягне певного мінімального розміру).

Крок 6: Завершення рекурсії. Коли розмір підпрямокутників став недостатнім для поділу, зупиніть рекурсію і позначте всі пусті клітинки як частину лабіринту, а всі решітки (внутрішні стіни) - як перешкоди.

# Опис розробки додатку

## **Опис структури проекту та його складових.**

**1.**Підключення необхідних просторів імен:



**2.**Оголошення класу розширень **Extensions**, який містить різні методи розширення для типу **IEnumerable<T>** і перерахування **CellState**.

**3.**Оголошення перерахування **CellState**, яке представляє стани комірки лабіринту, такі як верхня, права, нижня та ліва стінки, а також прапорець **Visited** та **Initial** для позначення відвіданих комірок і початкового стану комірки.

**4.**Оголошення структури **RemoveWallAction**, яка використовується для збереження інформації про сусідні комірки та збирання стінки, яка потрібно видалити.

**5.**Оголошення класу **Maze**, який включає поля і методи для створення і відображення лабіринту. Він має конструктор, який ініціалізує лабіринт заданого розміру та запускає метод **VisitCell** для розпочатку процесу автогенерації лабіринту. Також містить інші методи, такі як **GetNeighbours** для отримання сусідніх комірок, **VisitCell** для рекурсивного відвідування комірок і видалення стінок, а також **Display** для відображення лабіринту у вікні відладки.

**6.**Оголошення класу **Program**, який містить точку входу програми **Main**. В цьому методі створюється екземпляр класу **Maze** з розмірами 20x20 і викликається метод **Display** для відображення лабіринту.

## **Пояснення алгоритму гри та взаємодії з користувачем.**

Алгоритм автогенерації лабіринту, реалізований у даному коді, базується на алгоритмі "Рекурсивне ділення" (Recursive Division), який був описаний раніше.

Основні кроки алгоритму автогенерації лабіринту.

Створення об'єкту **Maze** з заданими розмірами (шириною і висотою).

У конструкторі **Maze** виконується початкова ініціалізація комірок лабіринту. Усі комірки початково встановлюються у стан **CellState.Initial**, що означає, що всі стінки присутні.

Виклик методу **VisitCell** з випадковими координатами початкової комірки, що починає процес автогенерації лабіринту.

У методі **VisitCell** комірка, яка передана як аргумент, помічається як відвідана, шляхом додавання флагу **CellState.Visited**.

Для кожної сусідньої комірки, яка ще не була відвідана, виконується наступне:

Видаляється стінка між поточною коміркою та сусідньою коміркою, шляхом віднімання відповідного флагу **CellState**.

Видаляється стінка у сусідній комірці, яка є протилежною до видаленої стінки, шляхом віднімання флагу **CellState** для протилежної стінки.

Рекурсивно викликається метод **VisitCell** для сусідньої комірки, що розширює процес автогенерації на нову комірку.

Процес продовжується рекурсивно, доки всі доступні комірки не будуть відвідані.

Після завершення автогенерації лабіринту, викликається метод **Display**, який відображає лабіринт у вікні відладки. Це робиться шляхом виведення рядків символів, які представляють стіни та проходи лабіринту.

Взаємодія з користувачем у даному коді обмежена. При запуску програми, створюється лабіринт заданого розміру, і він відображається у вікні відладки. Користувач може спостерігати згенерований лабіринт, але не може змінювати його або взаємодіяти з ним у реальному часі.

# Висновок

автогенерація лабіринтів є активною галуззю досліджень, особливо в контексті відеоігор, віртуальної реальності та комп'ютерних симуляцій. Дослідники використовують різні підходи, такі як генетичні алгоритми, клітинні автомати, алгоритми засновані на графах та нейромережі, для автоматичної генерації лабіринтів з різними характеристиками та рівнями складності.

Деякі дослідження можуть фокусуватися на ефективності алгоритмів генерації, шляхи покращення реалізації, збалансованості лабіринтів, різноманітності та інших аспектах. Деякі дослідження можуть також досліджувати, як генерування лабіринтів може вплинути на взаємодію гравців або вплинути на загальний геймплей відеоігор.

**Список викоритсаної літератури**

**1**.Shaker N., Abou-Zleikha M., Togelius J., Yannakakis G. N. (2016). "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO).

**2**.Smith G., Whitehead J. (2010). "Analyzing the Expressive Range of a Labyrinth Generating Level Generator." Proceedings of the 2010 Workshop on Procedural Content Generation in Games.

**3**.Dormans J. (2010). "Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games." Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).

**4**.Fernandez-Marquez J. L., Munoz-Avila H., Hervas-Martinez C. (2003). "A Knowledge-Based Method for Generating Dungeons in Computer Role-Playing Games." Proceedings of the 2003 AAAI Spring Symposium on Challenges to Game AI.

**5**.Mawhorter P., Mateas M., Jhala A. (2010). "Applying Grammatical Evolution to the Generation of Game Level Content." Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games.

6.Smith G., Mateas M. (2011). "Variations Forever: Flexibly Generating Rulesets from a Rule Template in the General Video Game AI Framework." Proceedings of the 2011 Workshop on Procedural Content Generation in Games.

**7**.Cardamone L., Lopes R., dos Santos Coelho L., Lanzi P. L., Banzhaf W. (2011). "Evolving Dungeon Maps for a Rogue-Like Game." Proceedings of the 2011 IEEE Congress on Evolutionary Computation.

Додаток

GitHub - https://github.com/illichCOVID/MazeCoursework