ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі хнуре

Дата звіту 6/12/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗм-23-3_Ковальов_М_В_скорочений
Автор
Ковальов Микита Віталійович
Науковий керівник / Експерт
Олена Олійник
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

25	5953	47108
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	3

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз		Колір тексту
ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://tit.dut.edu.ua/index.php/telecommunication/article/download/2509/2390/	12 0.20 %
2	https://library.vspu.net/bitstream/handle/123456789/9303/Tkachenko_Bak_2021.pdf?sequence=1&isAllowed=y	11 0.18 %
3	https://tit.dut.edu.ua/index.php/telecommunication/article/download/2509/2390/	10 0.17 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з Інтернету (0.55 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://tit.dut.edu.ua/index.php/telecommunication/article/download/2509/2390/	22 (2) 0.37 %
2	https://library.vspu.net/bitstream/handle/123456789/9303/Tkachenko_Bak_2021.pdf?sequence=1&isAllowed=y	11 (1) 0.18 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

ВСТУП

Процедурна генерація рівнів (PCG) є одним із ключових напрямків у сучасній розробці ігор, оскільки дозволяє створювати унікальні сценарії та локації без необхідності ручного проектування кожного елемента. Цей підхід сприяє не лише зниженню трудомісткості виробництва, але й підвищує реіграбельність продукту, відкриваючи нові можливості для творчості розробників.
За останні роки значну увагу отримали методи машинного навчання, що дозволяють адаптивно генерувати контент із врахуванням специфічних вимог до складності, балансу та логічної послідовності гри [1]. До того ж, інтеграція таких методів у процес розробки дає змогу оптимізувати роботу команд, заощаджуючи ресурси та скорочуючи час на тестування кінцевого продукту.
У цій кваліфікаційній роботі розглядаються сучасні підходи до застосування машинного навчання для генерації ігрових рівнів, із детальним аналізом переваг і недоліків кожного з них. Особлива увага приділяється алгоритмам навчання з підкріпленням, адже результати теоретичних досліджень свідчать про їхню високу ефективність для оптимізації процедурної генерації. Водночас, необхідно врахувати, що практична реалізація завжди включає опис теоретичних основ обраного підходу, що дозволить краще обґрунтувати прийняті рішення.
Метою даної роботи є розробка та апробація моделі генерації ігрових рівнів з використанням методів машинного навчання, де основним алгоритмом виступатиме Q-Learning. Проведення дослідження дозволить не лише порівняти ефективність, а й встановити потенціал для їх подальшого вдосконалення в контексті інтерактивних розважальних продуктів.

2

1 АНАЛІЗ ПРЕДМЕТОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ
1.1 Аналіз предметної галузі

Розробка комп'ютерних ігор на сьогодні є однією з найбільш динамічних та інноваційних сфер цифрових технологій. Вона охоплює широкий спектр технічних, дизайнерських і когнітивних завдань, серед яких важливе місце посідає формування ігрового середовища, зокрема генерація рівнів. У сучасних проєктах створення контенту ручними засобами стає дедалі менш ефективним через високі вимоги до масштабу, варіативності та персоналізації ігрового досвіду.
Відтак, індустрія все активніше звертається до процедурних підходів, які дозволяють автоматизувати створення ігрового контенту. Це, у свою чергу, ставить

ДОДАТОК Б

Програмний код генерації карти за допомогою алгоритму клітинного автомата

```
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using UnityEngine;
using UnityEngine.Tilemaps;
using Debug = UnityEngine.Debug;
using Random = System.Random;

public class CellularAutomataLevel : MonoBehaviour
{
    [Header("Map Settings")]
    public int width = 50;
    public int height = 50;
    [SerializeField] private int fillPercent = 45;
    [SerializeField] private int smoothingIterations = 5;
    [SerializeField] private int seed = 1111;

    [Header("Path Settings")]
    public Vector2Int start = new Vector2Int(1, 1);
    public Vector2Int end;
    private List<Vector2Int> validPositions;

    private Random rand;
    // 0 = wall, 1 = floor
    private int[,] map;

    [Header("Tilemap Settings")]
    public Tilemap tilemap;
    public TileBase floorTile;
    public TileBase wallTile;
    public TileBase exitTile;

    void Start()
    {
        validPositions = new List<Vector2Int>();
```

```
    end = new Vector2Int(width - 2, height - 2);
    RegenerateMap();
}

public Vector2Int FindValidFloorPosition()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (map[x, y] == 1)
            {
                return new Vector2Int(x, y);
            }
        }
    }
    return start;
}

public void RegenerateMap()
{
    Stopwatch sw = new Stopwatch();
    sw.Start();

    rand = new Random(seed);
    GenerateMap();
    for (int i = 0; i < smoothingIterations; i++)
    {
        SmoothMap();
    }

    EnsurePath();

    RemoveUnreachableZonesAndEnemies();

    DrawMap();
    sw.Stop();
    Debug.Log($"Map generated in {sw.ElapsedMilliseconds}ms");
    EventBroker.CallOnMapGenerated();
```

```
}

void DrawMap()
{
    tilemap.ClearAllTiles();
    validPositions.Clear();

    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            TileBase tile;

            if (map[x, y] == 1)
            {
                tile = floorTile;
                validPositions.Add(new Vector2Int(x, y));
            }
            else
            {
                tile = wallTile;
            }

            tilemap.SetTile(new Vector3Int(x, y, 0), tile);
        }
    }

    tilemap.SetTile(new Vector3Int(end.x, end.y, 0), exitTile);
}

void GenerateMap()
{
    map = new int[width, height];
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (x == 0 || y == 0 || x == width - 1 || y == height - 1)
                map[x, y] = 0;
```

```
                else
                    map[x, y] = (rand.Next(0, 100) < fillPercent) ? 1 : 0;
            }
        }
    }

    void SmoothMap()
    {
        int[,] newMap = new int[width, height];
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                int floorCount = CountFloorNeighbors(x, y);
                if (floorCount > 4)
                    newMap[x, y] = 1;
                else if (floorCount < 4)
                    newMap[x, y] = 0;
                else
                    newMap[x, y] = map[x, y];
            }
        }
        map = newMap;
    }

    public List<Vector2Int> GetValidPositions()
    {
        return validPositions;
    }

    int CountFloorNeighbors(int gridX, int gridY)
    {
        int count = 0;
        for (int neighborX = gridX - 1; neighborX <= gridX + 1;
neighborX++)
        {
            for (int neighborY = gridY - 1; neighborY <= gridY + 1;
neighborY++)
            {
```

```
                if (neighborX >= 0 && neighborX < width && neighborY >= 0
&& neighborY < height)
                {
                    if (neighborX != gridX || neighborY != gridY)
                        count += map[neighborX, neighborY];
                }
                else
                {
                    count++;
                }
            }
        }
        return count;
    }

    public Random GetRandom()
    {
        rand = new Random(seed);
        return rand;
    }

    void EnsurePath()
    {
        if (!PathExists(start, end))
        {
            CarvePath(start, end);
        }
    }

    public bool PathExists(Vector2Int from, Vector2Int to)
    {
        bool[][] visited = new bool[width][];
        for (int i = 0; i < width; i++)
        {
            visited[i] = new bool[height];
        }

        Queue<Vector2Int> queue = new Queue<Vector2Int>();
        queue.Enqueue(from);
```

```
        visited[from.x][from.y] = true;

        while (queue.Count > 0)
        {
            Vector2Int current = queue.Dequeue();
            if (current == to)
                return true;

            foreach (Vector2Int dir in Directions())
            {
                Vector2Int neighbor = current + dir;
                if (InBounds(neighbor) && !visited[neighbor.x][neighbor.y]
&& map[neighbor.x, neighbor.y] == 1)
                {
                    visited[neighbor.x][neighbor.y] = true;
                    queue.Enqueue(neighbor);
                }
            }
        }
        return false;
    }

    void CarvePath(Vector2Int from, Vector2Int to)
    {
        Vector2Int current = from;
        int maxSteps = width * height;
        int steps = 0;

        while (current != to && steps < maxSteps)
        {
            map[current.x, current.y] = 1;
            List<Vector2Int> possibleMoves = new List<Vector2Int>();

            if (current.x < to.x)
                possibleMoves.Add(new Vector2Int(current.x + 1,
current.y));
            if (current.x > to.x)
                possibleMoves.Add(new Vector2Int(current.x - 1,
current.y));
```

```
            if (current.y < to.y)
                possibleMoves.Add(new Vector2Int(current.x, current.y +
1));
            if (current.y > to.y)
                possibleMoves.Add(new Vector2Int(current.x, current.y -
1));

            possibleMoves.AddRange(new Vector2Int[]
            {
                new Vector2Int(current.x + 1, current.y),
                new Vector2Int(current.x - 1, current.y),
                new Vector2Int(current.x, current.y + 1),
                new Vector2Int(current.x, current.y - 1)
            });
            possibleMoves = possibleMoves.OrderBy(_ =>
rand.Next()).ToList();

            bool moved = false;
            foreach (Vector2Int move in possibleMoves)
            {
                if (InBounds(move) && move != from)
                {
                    current = move;
                    moved = true;
                    break;
                }
            }

            if (!moved)
                break;
            steps++;
        }

        map[to.x, to.y] = 1;
    }

    public int[,] ComputeDijkstraMap(Vector2Int source)
    {
        int[,] distance = new int[width, height];
```

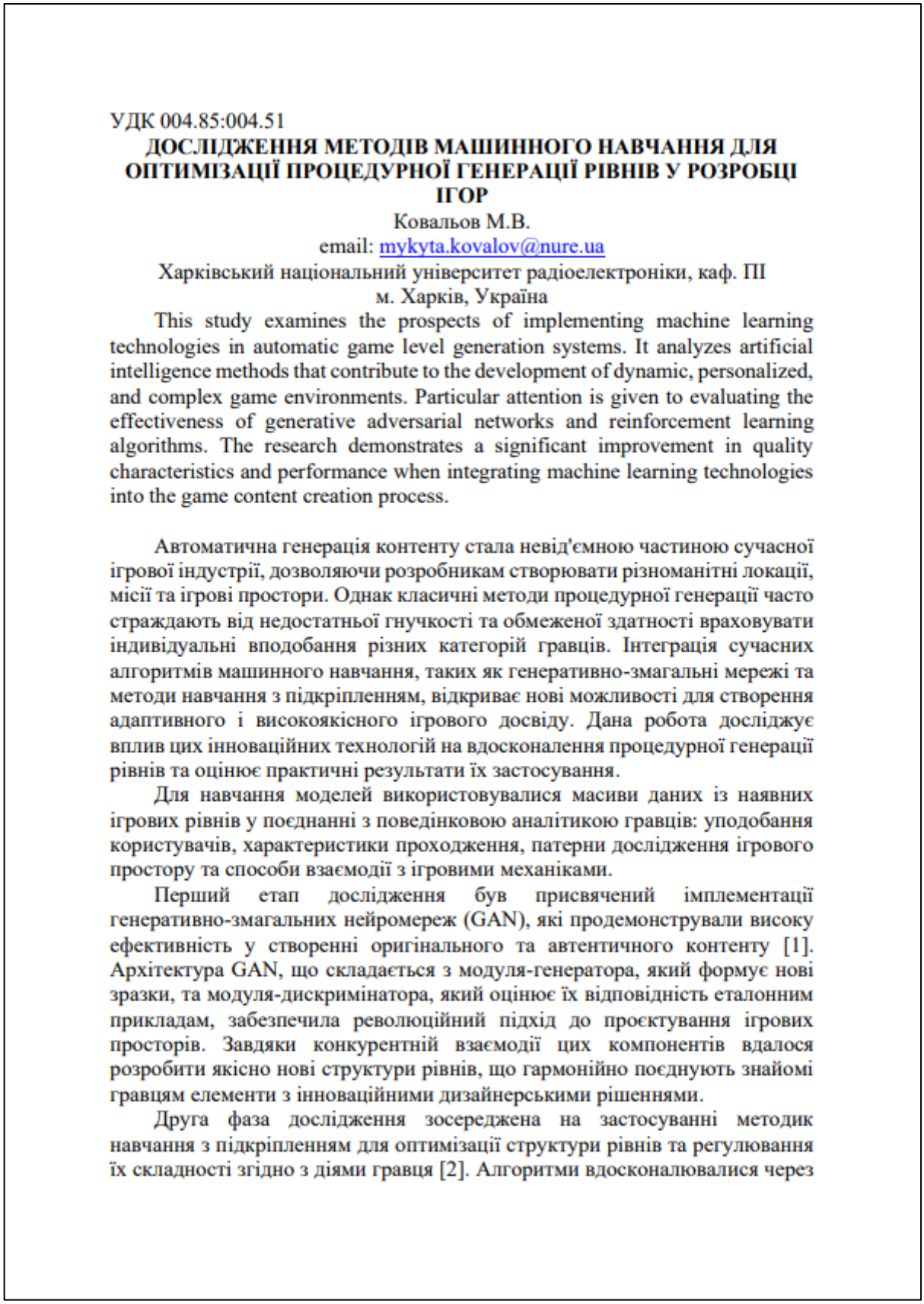
```
    for (int x = 0; x < width; x++)
        for (int y = 0; y < height; y++)
            distance[x, y] = int.MaxValue;

    distance[source.x, source.y] = 0;
    Queue<Vector2Int> queue = new Queue<Vector2Int>();
    queue.Enqueue(source);

    while (queue.Count > 0)
    {
        Vector2Int current = queue.Dequeue();
        int currentDist = distance[current.x, current.y];
        foreach (Vector2Int dir in Directions())
        {
            Vector2Int neighbor = current + dir;
            if (InBounds(neighbor) && map[neighbor.x, neighbor.y] == 1)
            {
                int newDist = currentDist + 1;
                if (newDist < distance[neighbor.x, neighbor.y])
                {
                    distance[neighbor.x, neighbor.y] = newDist;
                    queue.Enqueue(neighbor);
                }
            }
        }
    }
    return distance;
}

void RemoveSentinelPoints(int[,] dijkstraMap)
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (map[x, y] == 1 && dijkstraMap[x, y] == int.MaxValue)
            {
                map[x, y] = 0;
            }
```

```
            }
        }
    }

    public void RemoveUnreachableZonesAndEnemies()
    {
        int[,] dijkstraMap = ComputeDijkstraMap(start);
        RemoveSentinelPoints(dijkstraMap);
        DrawMap();
        var enemies = GameObject.FindGameObjectsWithTag("Enemy");
        foreach (var enemy in enemies)
        {
            Vector3 pos = enemy.transform.position;
            Vector2Int gridPos = new Vector2Int(Mathf.FloorToInt(pos.x),
Mathf.FloorToInt(pos.y));

            if (!InBounds(gridPos) || map[gridPos.x, gridPos.y] == 0)
            {
                Destroy(enemy);
            }
        }
    }

    public bool InBounds(Vector2Int pos)
    {
        return pos.x >= 0 && pos.x < width && pos.y >= 0 && pos.y < height;
    }

    public int[,] GetMap()
    {
        return map;
    }

    List<Vector2Int> Directions()
    {
        return new List<Vector2Int> {
            new Vector2Int(1, 0),
            new Vector2Int(-1, 0),
            new Vector2Int(0, 1),
```

```
            new Vector2Int(0, -1)
        };
    }

    public void SmoothMapExternally()
    {
        SmoothMap();
        DrawMap();
        Debug.Log("SmoothMapExternally executed.");
    }


    public void RandomWallRemoveExternally()
    {
        int[,] currentMap = GetMap();
        if (currentMap == null) return;

        for (int x = 1; x < width - 1; x++)
        {
            for (int y = 1; y < height - 1; y++)
            {
                if (currentMap[x, y] == 0 && rand.NextDouble() < 0.05)
                {
                    currentMap[x, y] = 1;
                }
            }
        }
        DrawMap();
        Debug.Log("RandomWallRemoveExternally executed.");
    }

    public void ForcePath()
    {
        EnsurePath();
        DrawMap();
        Debug.Log("ForcePath executed.");
    }
}
```

ДОДАТОК В

Апробація результатів роботи



УДК 004.85:004.51

ДОСЛІДЖЕННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ РІВНІВ У РОЗРОБЦІ ІГОР

Ковальов М.В.
email: mykyta.kovalov@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study examines the prospects of implementing machine learning technologies in automatic game level generation systems. It analyzes artificial intelligence methods that contribute to the development of dynamic, personalized, and complex game environments. Particular attention is given to evaluating the effectiveness of generative adversarial networks and reinforcement learning algorithms. The research demonstrates a significant improvement in quality characteristics and performance when integrating machine learning technologies into the game content creation process.

Автоматична генерація контенту стала невід'ємною частиною сучасної ігрової індустрії, дозволяючи розробникам створювати різноманітні локації, місії та ігрові простори. Однак класичні методи процедурної генерації часто страждають від недостатньої гнучкості та обмеженої здатності враховувати індивідуальні вподобання різних категорій гравців. Інтеграція сучасних алгоритмів машинного навчання, таких як генеративно-змагальні мережі та методи навчання з підкріпленням, відкриває нові можливості для створення адаптивного і високоякісного ігрового досвіду. Дана робота досліджує вплив цих інноваційних технологій на вдосконалення процедурної генерації рівнів та оцінює практичні результати їх застосування.

Для навчання моделей використовувалися масиви даних із наявних ігрових рівнів у поєднанні з поведінковою аналітикою гравців: уподобання користувачів, характеристики проходження, патерни дослідження ігрового простору та способи взаємодії з ігровими механіками.

Перший етап дослідження був присвячений імплементації генеративно-змагальних нейромереж (GAN), які продемонстрували високу ефективність у створенні оригінального та автентичного контенту [1]. Архітектура GAN, що складається з модуля-генератора, який формує нові зразки, та модуля-дискримінатора, який оцінює їх відповідність еталонним прикладам, забезпечила революційний підхід до проєктування ігрових просторів. Завдяки конкурентній взаємодії цих компонентів вдалося розробити якісно нові структури рівнів, що гармонійно поєднують знайомі гравцям елементи з інноваційними дизайнерськими рішеннями.

Друга фаза дослідження зосереджена на застосуванні методик навчання з підкріпленням для оптимізації структури рівнів та регулювання їх складності згідно з діями гравця [2]. Алгоритми вдосконалювалися через

Рисунок Б.1 – Перша сторінка матеріалу

взаємодію з віртуальним середовищем, отримуючи винагороди за досягнення таких параметрів, як збалансованість геймплею, оптимальний рівень складності та плавна прогресія навичок. На практиці система могла автоматично коригувати щільність противників або розташування ресурсів залежно від стилю проходження гравцем попередніх локацій. На рисунку 1 видно, що агент взаємодіє із середовищем, отримуючи від нього інформацію про стан та винагороду, що дозволяє алгоритму адаптуватися до гравця.

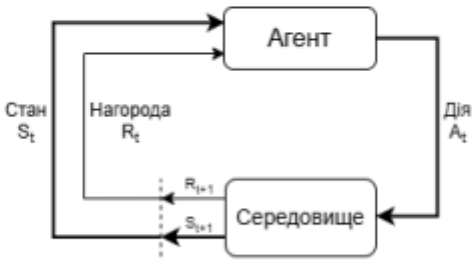


Рисунок 1. Схема взаємодії агента з середовищем у навчанні з підкріпленням

Експериментальне тестування переконливо демонструє, що синергія описаних методик значно підвищує якість автоматично генерованого контенту порівняно з традиційними алгоритмами. Подібний підхід узгоджується з результатами досліджень у сфері гібридних імітаційно-керуючих моделей, які дозволяють покращити процес прийняття рішень в умовах часткової невизначеності [3]. Отримані результати відкривають широкі перспективи для впровадження подібних технологій у комерційну розробку ігор, дозволяючи створювати віртуальні простори, які не лише відповідають сучасним індустріальним стандартам, а й формують нові тенденції у ігровому дизайні.

Список використаних джерел:

1. M. Jiang and L. Zhang, "An Interactive Evolution Strategy based Deep Convolutional Generative Adversarial Network for 2D Video Game Level Procedural Content Generation," 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-6, doi: 10.1109/IJCNN52387.2021.9533847.

2. A. Servat and H. S. Mohamadi, "Immersive Game Worlds: Using Deep Reinforcement Learning for Lifelike Non-Player Characters," 2023 International Serious Games Symposium (ISGS), Tehran, Iran, Islamic Republic of, 2023, pp. 1-5, doi: 10.1109/ISGS61252.2023.10559740.

3. Filatov V. O.; Yerokhin A. L.; Zolotukhin O. V.; Kudryavtseva M. S. Hybrid simulation models for complex decision-making problems with partial uncertainty. Information extraction and processing 2022-12-19 | Journal article, doi: 10.15407/vidbir2022.50.078

Рисунок Б.2 – Друга сторінка матеріалу

Додаток Г

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Дослідження методів машинного навчання для оптимізації процедурної генерації рівнів у розробці ігор

Ковальов Микита Віталійович, ІПЗм-23-3
Науковий керівник: проф. Шубін І.Ю.

20 червня 2025

Дослідження

Актуальність та стан розвитку галузі:

У сучасній розробці ігор процедурна генерація рівнів (PCG) дозволяє автоматизувати створення контенту, підвищити реіграбельність та зменшити витрати. Проте традиційні методи обмежені статичними правилами і не враховують динаміку геймплею. Методи машинного навчання, зокрема навчання з підкріпленням, дають змогу створювати адаптивні рівні, які враховують змінні умови та поведінку гравця.

Напрям дослідження:

Розробка та дослідження моделі процедурної генерації рівнів з використанням алгоритму Q-Learning у середовищі Unity.

Об'єкт дослідження:

Методи машинного навчання для процедурної генерації рівнів у комп'ютерних іграх.

2

Огляд літератури (аналогів)

У галузі процедурної генерації рівнів широко використовуються класичні алгоритми (Wave Function Collapse, Cellular Automata, Marching Squares), а також сучасні підходи на основі машинного навчання: нейронні мережі (CNN), генеративно-змагальні мережі (GAN), навчання з підкріпленням (RL). Серед яких:

- **Класичні методи генерації** (клітинкові автомати, генерація за шумами, Wave Function Collapse) дозволяють створювати рівні на основі заздалегідь заданих правил. Вони прості у реалізації та добре керовані, однак часто обмежені в гнучкості, масштабованості та адаптивності до гравця.
- **Методи машинного навчання** — зокрема, нейронні мережі, GAN (генеративно-змагальні мережі), автоенкодери — мають потенціал до створення більш складного та адаптивного контенту. Вони здатні виявляти шаблони, навчатись на прикладах та створювати нові рівні, подібні до тих, що використовувалися під час навчання.
- **Порівняльний аналіз** показує, що ML-методи переважають у завданнях, де важлива непередбачуваність, глибина варіацій та відповідність геймплейним очікуванням. Водночас вони вимагають більше ресурсів, знань і часу на підготовку даних.

Більшість досліджень у сфері RL для генерації рівнів зосереджуються на концептуальних моделях або прикладах без практичної реалізації у геймдевелопмент-движках. Недостатньо вивчені аспекти інтеграції RL (зокрема Q-Learning) з конкретними середовищами на кшталт Unity, а також формалізація станів, функцій винагород і адаптації до реальних ігрових метрик.

Постановка задачі

Існуючі методи процедурної генерації рівнів не забезпечують достатньої гнучкості, адаптивності та ігрової доцільності. Більшість з них не враховують динаміку гравця і потребують ручного налаштування. Водночас застосування алгоритмів навчання з підкріпленням у цій сфері недостатньо досліджене з точки зору практичної реалізації у реальному ігровому середовищі (Unity).

Реалізація наукового дослідження скаладається з наступних етапів:

- Побудова моделі генерації рівнів на основі Q-Learning.
- Реалізація цієї моделі у Unity з використанням C#.
- Формування та застосування функції винагороди для навчання агента.
- Отримання адаптивних, прохідних і збалансованих ігрових рівнів.
- Аналіз ефективності підходу та визначення його обмежень.

Методологія

Для проведення експериментального дослідження було реалізовано систему процедурної генерації рівнів у середовищі Unity з агентом, що навчається за алгоритмом Q-Learning. Генератор створює рівні у вигляді двовимірної сітки на основі клітинного автомата, а агент модифікує їх шляхом послідовних дій.

Методологічні принципи дослідження:

— Одинакова початкова структура, всі рівні ініціалізуються однаковим базовим алгоритмом, що забезпечує порівнюваність результатів.

— Контроль середовища навчання агента відбувається в симульованих, стабільних умовах з фіксованими параметрами (розмір карти, набір дій, функція винагороди).

— Вимірювані метрики, під час навчання фіксуються ключові параметри рівня (прохідність, щільність, кількість об'єктів), які використовуються для оцінки ефективності.

— Статистичний підхід, де результати аналізуються за серією запусків (епізодів), що дозволяє виявити закономірності навчання агента та стабільність моделі.

— Використання Python для обробки згенерованих рівнів та візуалізації результатів.

Архітектура система для проведення експериментального дослідження

Розроблена система складається з модулів, що взаємодіють у процесі процедурної генерації рівня та навчання агента:

— Генератор початкового рівня створює базову структуру карти за допомогою клітинного автомата.

— Агент Q-Learning визначає стан рівня, обирає дію згідно з ε-жадібною політикою, оновлює Q-таблицю на основі винагороди.

— Оцінювач стану (State Metrics) розраховує прохідність рівня, кількість ворогів, скарбів, щільність прохідних клітин.

— Блок можливих дій формує перелік допустимих змін (видалення стін, розміщення ворогів, згладжування тощо).

— Функція винагороди аналізує новий стан і обчислює винагороду на основі критеріїв якості (баланс, прохідність, різноманіття).

— Модуль візуалізації відображає карту в Unity, дозволяє перевірити рівень вручну або автоматично (через pathfinding).

— Python-аналітика Застосовується для збереження статистики, побудови графіків динаміки навчання та аналізу результатів.

Опис програмного забезпечення, що було використано у дослідженні

Для проведення експериментального дослідження ефективності застосування алгоритму Q-Learning у задачі процедурної генерації ігрових рівнів було використано відповідне програмне забезпечення, що забезпечує повний цикл моделювання, симуляції та аналізу. В якості основного середовища розробки використано:

- **Unity Editor (версія 2022.3.35f1)** - як платформа для створення, візуалізації та тестування рівнів.

Основні інструменти та мови:

- **C# (.NET Standard 2.1)** - для реалізації агентів, логіки генерації, Q-таблиці та взаємодії зі середовищем.
- **Python 3.11** - для статистичного аналізу та побудови графіків.
- **Matplotlib, Pandas** - бібліотеки для обробки й візуалізації результатів.
- **Unity Tilemap API** - для побудови двовимірних рівнів у форматі сітки.
- **A Pathfinding (Unity)** - для перевірки наявності прохідного маршруту на згенерованій карті.

Обраний стек технологій дозволив повністю реалізувати цикл генерації, навчання агента та аналіза якості результатів.

Зміст проведеного експерименту

Для оцінки ефективності роботи агента, що навчається за алгоритмом Q-Learning, було побудовано послідовну процедуру вимірювання якості згенерованих рівнів та динаміки навчання.

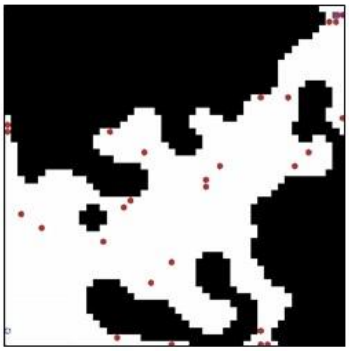
Основні етапи:

- запуск системи у середовищі Unity з фіксованими початковими умовами (розмір карти, структура, набір дій);
- проведення 200 епізодів навчання з автоматичним логуванням ключових показників на кожному кроці;
- збереження результатів навчання у CSV-форматі;
- обробка та агрегація даних у Python за допомогою Pandas;
- побудова графіків зміни показників (кількість ворогів на рівні, відсоток клітин підлоги, кількість скарбів на рівні) за допомогою Matplotlib;
- повторення експерименту 3 рази для перевірки стабільності результатів;

Окремо оцінено вплив різних гіперпараметрів (ε, α, γ) на швидкість і якість навчання агента.

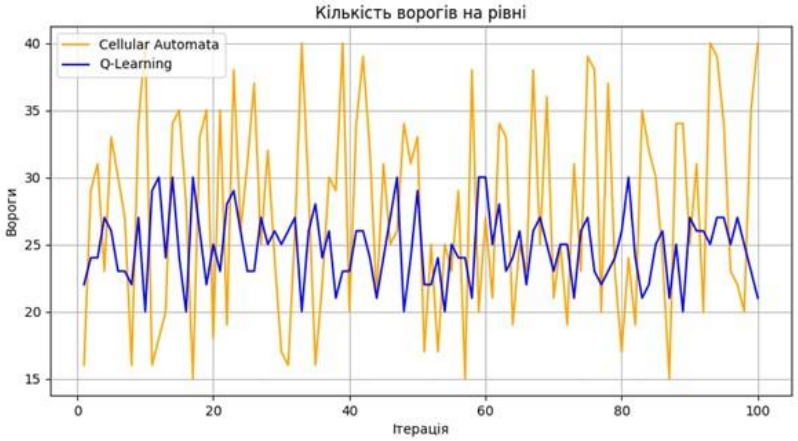
Результати експерименту

Cellular Automata



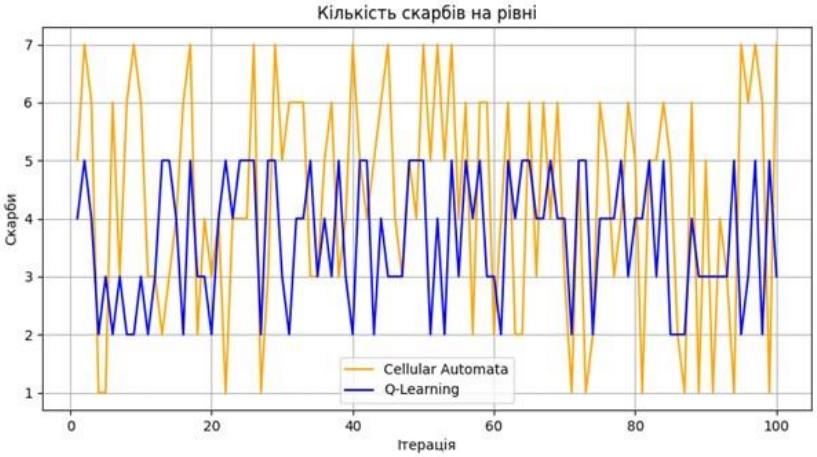
Cellular Automata + Q-Learning

Результати експерименту



Результати експерименту

Результати експерименту



Аналіз отриманих результатів

Метою дослідження була розробка та апробація моделі процедурної генерації рівнів із використанням Q-Learning. Реалізована система дозволила досягти цієї мети, агент успішно навчався наданим правилам і модифікував рівні згідно з заданими критеріями якості.

У процесі навчання було досягнуто зростання середньої винагороди, зменшення кількості кроків до прохідного рівня, а також зростання стабільності результатів. Дані свідчать про здатність агента адаптувати свою поведінку до поставлених цілей.

Q-Learning продемонстрував ефективність у задачі покрокової оптимізації ігрового рівня на основі зворотного зв'язку. Найкращі результати отримано при балансі між дослідженням середовища (exploration) та експлуатацією знань (exploitation).

Результати підтверджують доцільність застосування навчання з підкріпленням у контексті генерації рівнів. Робота демонструє практичну реалізацію теоретичних моделей RL у реальному геймдев-середовищі, що розширює можливості адаптивного геймдизайну.

Публікація результатів


ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ПОЧЕСНА ГРАМОТА

НАГОРОДЖУЄТЬСЯ

студент КОВАЛЬОВ МИКИТА

ХНУРЕ, каф. ІП

за активну участь

у 29-му Міжнародному молодіжному форумі

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

з доповіддю «ДОСЛІДЖЕННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ РІВНІВ У РОЗРОБЦІ ІГОР»

В.о. ректора Юрій РОМАНЕНКОВ

17.04.2025 р.

Висновки

Отримані результати демонструють реалістичність та прикладну корисність обраного підходу: агент на основі Q-Learning здатен покроково покращувати структуру рівня відповідно до заданих метрик, забезпечуючи прохідність та ігрову логіку. Реалізація у середовищі Unity підтвердила технічну здійсненність і практичну придатність моделі для задач геймдизайну.

Розроблений підхід може бути адаптований для більш складних ігрових сценаріїв або масштабованих структур. У майбутніх дослідженнях доцільно розглянути:

- **Graph Q-Learning** - для роботи з рівнями, представленими як графи, а не сітки;
- **Deep Q-Networks (DQN)** - для навчання в середовищах із великою кількістю можливих станів без явного Q-табличного подання.

Це відкриває перспективу для створення більш гнучких, контекстно-залежних генераторів рівнів, що здатні адаптуватися до складніших ігрових ситуацій у реальному часі.

ДОДАТОК Ґ

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

1

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-3 (група)

Микита КОВАЛЬОВ
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

У рисунках необхідно зазначити джерело походження.
13.06.2025