					ІТ-42.23 1153.01 ПЗ		
Зм.	Лист	№ докум.	Підп.	Дата			
Розроб.						Літ.	Лист	Листів
Розроб.							7	88
						КПІ ім. Ігоря Сікорського каф. ТК Гр. ІТ-42		
Н. контр.								
Затв.								

ПЕРЕЛІК СКОРОЧЕНЬ

БД – База даних

СУБД – Система управління базами даних

ПЗ – Програмне забезпечення

HTTP – HyperText Transfer Protocol

HTML – HyperText Markup Language

URL – Uniform Resource Locator

CSS – Cascading Style Sheets

IIS – Internet Information Server

AIC – Автоматизована інформаційна система

MVC – Model-View-Controller

IT – Інформаційні технології

ЕОМ – Електронна обчислювальна машина

API – Application Programming Interface

CLR – Common Language Runtime

CIL – Common Intermediate Language

SQL – Structured query language

EF – Entity Framework

ORM – Object-relational mapping

XML – Extensible Markup Language

CMS – Content Management System

LINQ – Language Integrated Query

ДСН – Державні санітарні правила та норми

ДБН – Державні будівельні норми

ВСТУП

В сучасному світі ігри є двигоном прогрессу комп’ютерної індустрії. Фінансовий оборот ігрової індустрії зрстає з кожним роком. Згідно з цим розробники намагаються запропонувати все максимально якісні та захоплюючи ігри. Однією з важливих особливостей багатьох ігрових компаній є їхні пропрацьовані до найтонших деталей ігрові світи. Великі компанії витрачають сотні тисяч та наймають сотні працівників які займається однією справою – створюють ігрові карти та рівні для нового проекту. Цей процес часто дуже трудомісткий та потребує не один рік роботи. Ці фахівці мають попрацювати з кожною найменшою частинкою яка знаходиться в межах рівня, будь-то травинка або камінь. Останнім часом, як альтернативу такому довгому процесу використовують алгоритми процедурної генерації рівнів. Вони часто є досить популярними серед невеликих компаній, котрі складаються всього з декілька чоловік, та не можуть собі дозволити такий маштаб проробки рівнів гри. Також і великі компанії використовують генерацію рівнів, жертвуючи прискіпливой розробкой кожного лементу на рівня, але натомість вони отримують потужній засіб генерації, що дозволить програмі кожне проходження генерувати зовсім новий рівень.

Існуючі алгоритми процедурної генерації розробляються виключно під конкретну гру, зазвичай їх не можливо застосувати до інших ігор, а модуль штучного інтелекту для визначення вподобання гравця робить алгоритм менш випадковим та більш налаштованим на конкретного гравця. Враховуючи вищезазначену ситуацію створення модулю процедурної генерації з елементами штучного інтелекту є актуальним напрямком розробки.

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ, АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ФОРМУВАННЯ ВИМОГ

1.1 Теоретичні відомості

Процедурна генерація (англ. Procedural generation, скор. PCG) — автоматичне створення ігрового контенту за допомогою алгоритмів. Іншими словами, процедурна генерація представляє собою програмне забезпечення, яке може створювати ігровий контент самостійно, або спільно при взаємодії з гравцями або геймдизайнером.

Під контентом розуміється створення рівнів гри, карти ігрового світу, правил гри, текстур, сюжетів, предметів, квестів, музики, зброї, транспортних засобів, персонажів і іншого. В даному контексті під іграми розуміються комп'ютерні ігри, відеоігри, настільні ігри, ігри в карти, головоломки та інші[1].

Ключовою особливістю створюваного контенту є те, що він повинен бути іграбельним — гравець повинен бути в змозі пройти створений рівень, використовувати сгенерированное зброю, піднятися по згенерованої сходах тощо.

Терміни «процедурний» і «генерація» відносяться до комп'ютерів, так як процедурна генерація має бути запущена у вигляді процедури на комп'ютері, яка видасть на вихід щось для подальшого застосування. При цьому процедура є частиною чогось більшого, наприклад всієї гри або інструменту геймдизайнера[1].

Генеруємий контент повинен відповідати певним вимогам і таким чином вирішувати відповідні проблеми, і на практиці найчастіше розглядаються наступні властивості процедурної генерації:

- Швидкість — в залежності від завдання вимоги різняться від мілісекунд до місяців, але в загальному випадку контент повинен бути створений вчасно для задоволення потреб ігрового процесу.

- Надійність — один з найважливіших факторів якісно згенерованого контенту, є гарантування виконання заданих критеріїв для генерації, а саме, контент обов’язково має бути іграбельним, наприклад завжди забезпечувати можливість проходження гравцем лабіринту, тощо.
- Різноманітність — процедурна генерація повинна давати змогу створювати різноманітний, різнобарвний контент, який не викликав би у гравця відчуття одноманітності.
- Креативність і правдоподібність — згенерований контент має бути подібним до контенту створеного людиною, а не генератором.

Метою використання процедурної генерації може бути створення ігрового контенту без участі людини (що може бути як менш затратно, так і допомагати геймдизайнеру у вирішенні їхніх завдань), розробка інших типів ігор (поліпшення показників різноманітності і реіграбельності), адаптація ігор під гравця «на льоту», поліпшення контенту за допомогою алгоритмічних рішень та інше.

Процедурна генерація це ефективний інструмент для вирішення проблем. Цей механізм задіюється для створення безлічі рівнів в необмеженому середовищі або для додавання бажаних елементів в дизайн персонажу - так вони не будуть виглядати і діяти ординарно. Обробляти таку інформацію вручну досить важко, причому на це піде чимало часу. Мінусом подібної системи, є те, що на написання і тестування алгаротимів також потрібна велика кількість часу та праці, особливо, якщо передбачається їх взаємодія з іншими системами[2].

1.2 Огляд існуючих рішень

Процедурна генерація широко використовується в сучасній ігровій промисловості. Багато ігрових компаній використовують генерацію для того, щоб додати більше контенту та поліпшити ігровий процесс.

Генерація використовується для ігор різних жанрів та різного рівня, від невеликих інді-ігор до ААА проектів. Зазвичай в загальний доступ викладається лише концепція алгоритмів використанних для процедурної генерації, а самі алгоритми розроблюються заново для кожної конкретної гри та можуть бути повністю використанні лише в рамках конкретної гри.

Було проведено огляд існуючих систем процедурної генерації, здійснено аналіз складових. Для порівняння були відібрані наступні алгоритми: процедурна генерація рівнів для гри M.E.R.C та процедурна генерація світу в Minecraft.

Рисунок 1.1 – M.E.R.C.

Алгоритм процедурної генерації M.E.R.C.

M.E.R.C. – це симулятор тактичного бою в реальному часі, де гравець одночасно бере під своє керівництво загін з чотирьох бійців[3].

Особливості процедурної генерації данного проекту:

- Рівень повинен мати один основний маршрут проходження;

- Рівень повинен містити від одного до трьох тупикових шляхів для виконання побічних цілей;
- Рівень повинен містити від одного до трьох коротких шляхів, які будуть використовуватись як альтернативні маршрути;
- Алгоритм має генерувати випадкові об’єкти інтер’єру і укриттів в кожному фрагменті рівня;
- Алгоритм має генерувати ворогів на підставі «графіку темпу»;
- Алгоритм має генерувати різні типи та рівні ворогів на підставі складності конкретної місії;
- Алгоритм повинен бути детермінованим і генерувати абсолютно однакові рівні для спільної гри по мережі.

Для спрощення проектування і розробки алгоритму було вирішено використати систему розбивки основної задачі на окремі підзадачі. Генеруємий рівень сприймався як створюваний в кілька єтапів, кожен з яких додає рівню новий прошарок складності та вирішує одну або декілька підзадач.

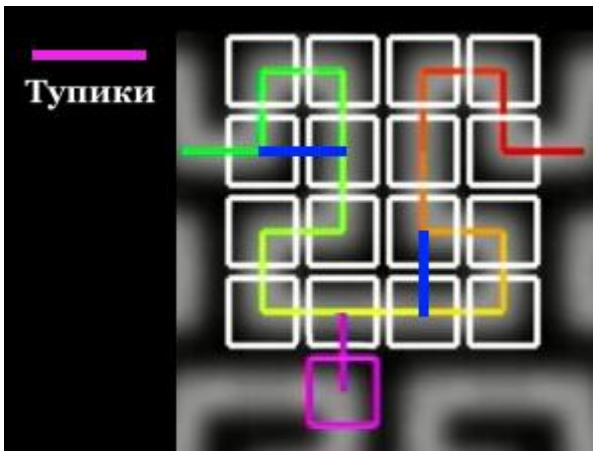


Рисунок 1.2 – Структура маршруту рівня

Процес починався з базового рівня – структури (маршруту) яка зображена на рис. 1.2. На цьому єтапі генерувалась карта відповідних розмірів, розбивалася на сектори та на основі цього будувався основний маршрут.

Наступним єтапом була оцінка згенерованих секторів та маршруту для пошуку найбільш близьких ділянок для формування коротких шляхів

(альтернативних маршрутів) та випадковим чином додаються побічні шляхи з тупиками. Тупики створюються в місцях, де немає секторів карти.

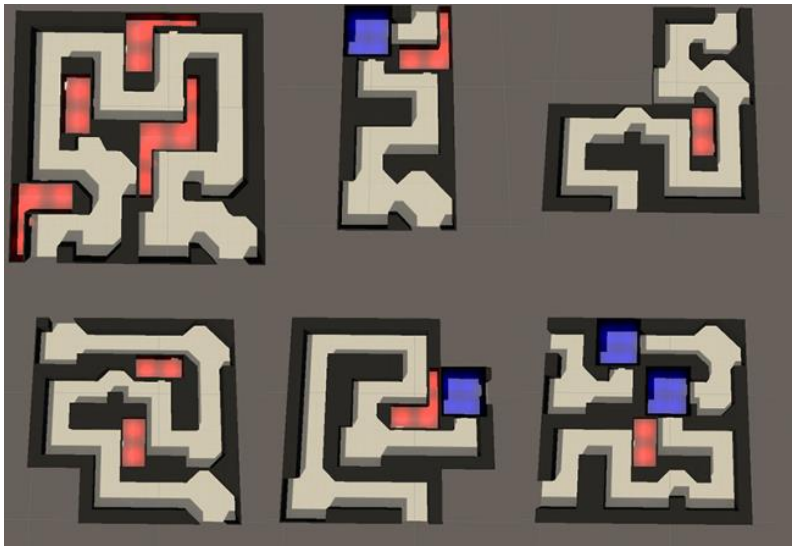


Рисунок 1.3 – Згенеровані різні типи рівнів

Після генерації основних, тупикових та коротких (альтернативних) маршрутів, карта розбивається на основі секторів, та кожен окремий сектор передається дизайнеру рівнів, який створює різні версії одного фрагменту рівня[3].

При генеруванні рівня система випадково вибирає одну з варіацій кожного фрагмента рівня, забезпечуючи більшу візуальну різноманітність(рис. 1.3).

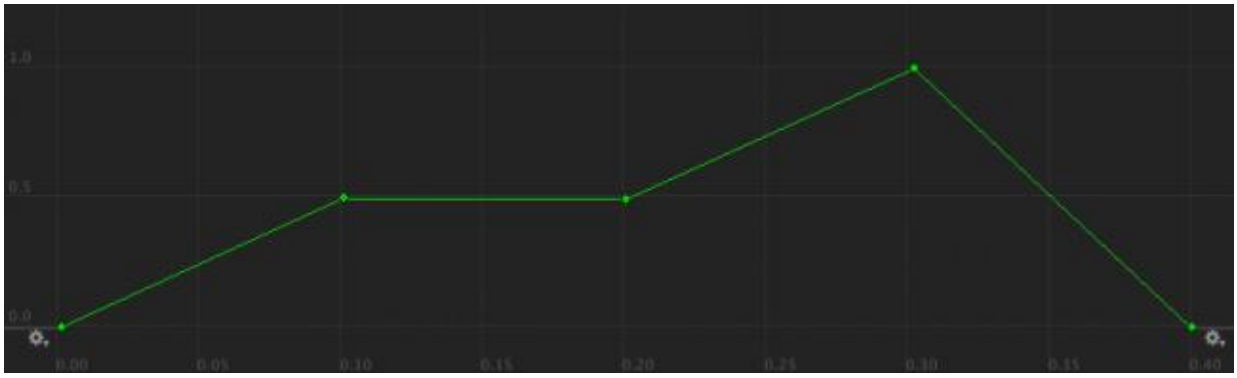


Рисунок 1.4 – Графік темпу

Останнім єтапом пілся генерації структури та основних об’єктів інтерфейсу є генерація випадкових укриттів в випадкових місцях в межах сектору, та створення різних за типом та рівнем ворогів на підставі «графіку темпу»(рис. 1.4). Поняттю «графік темпу» відповідає випадково згенерована крива, точки якої відповідають окремим секторам рівня. Від

позиції точки сектора на графіку залежить інтенсивність появи ворогів у цьому секторі.

Рисунок 1.5 – Згенерований рівень

Почавши зі створення основного маршруту та накладаючи прошарки з кожним єтапом з урахуванням вимог, була побудована досить надійна система, яку можна регулювати та надбудовувати в процесі розробки(рис. 1.5).

Серед недоліків можна виділити повну лінійність створюваних рівнів, та низьку варіативність дизайну окремих сегментів, що може призвести до появи однакових сегментів в рамках навіть всього одного рівня.

Також, серед недоліків, можна виділити генерацію кривої «графіку темпу», крива може згенеруватися з частим поверненням в точку мінімуму, що призведе низьку інтенсивность появлення ворогів і зменшення цікавості данного рівня для гравця.

Рисунок 1.6 – Minecraft

Алгоритм процедурної генерації Minecraft

Minecraft — комп'ютерна інді-гра в жанрі пісочниці з елементами симулятора виживання і відкритим світом. Minecraft надає в розпорядження гравця тривимірний процедурно генеруємий світ, що повністю складається з кубічних блоків, для цього використовуються алгоритми генерації карти[4].

Генерація карти — це процес випадкового створення географічних і геологічних об'єктів на карті при першому запуску гри на порожньому слоті для ігрового світу[4].

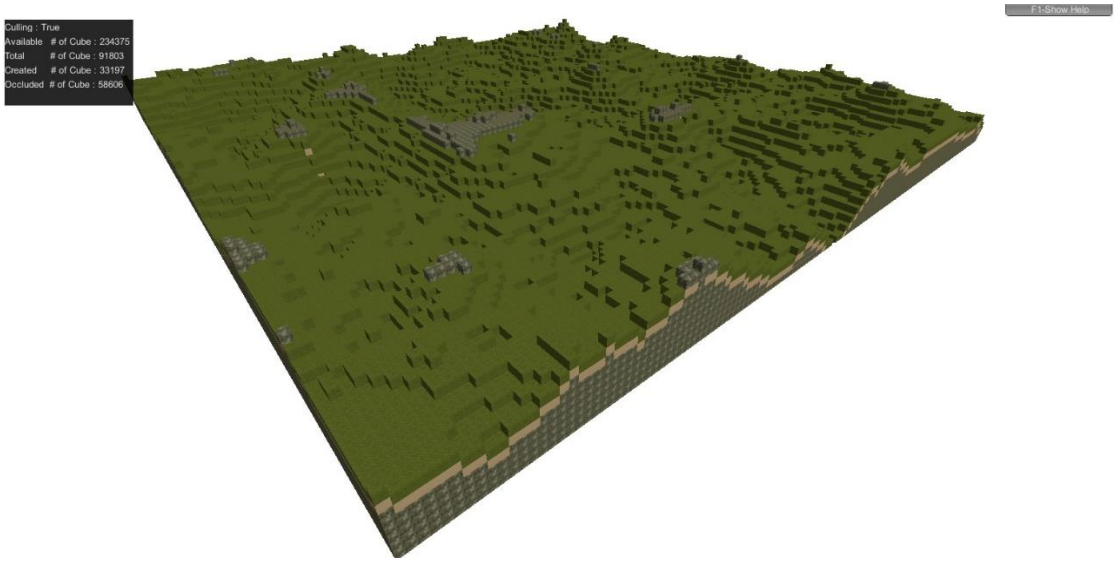


Рисунок 1.7 – Фрагмент блоку

При генеруванні ландшафту створюються фрагменти 16*16*16 блоків(рис. 1.7), саме з множини цих фрагментів і буде складатися світ. Для зберігання і завантаження створеного світу кожному фрагменту присвоюється значення зсуву, яке зберігається у вигляді 32-бітного цілого числа.

Світ в Minecraft генерується автоматично під час гри, коли гравець підходить до краю згенерованої зони, запускається алгоритм процедурної генерації і створюється наступна зона, що приєднується до краю старої.

Для генерації використовується система що генерує «висоту землі» та «щільність», всі блоки, щільність яких, меньше ніж нулю стають повітрям,

а блоки, з щільністю більше або рівною нулю перетворюються в землю. Для того щоб нижній шар був твердим, а верхній — ні, до згенерованого значення щільності додається значення висоти.

Рисунок 1.8 – Згенерована карта

Дана система має очевидні недоліки, один з основних, це недолік в продуктивності, так як при генерації проводиться величезна кількість розрахунків. Проблеми з продуктивністю призводять до того, що в момент генерації наступного сектору, який виникає під час гри,

процедурна генерація споживає всі доступні ресурси системи, що зумовлює винекнення так званих «фрізів» під час гри.
Іншим недоліком є те, що світ, за ціми алгоритмами, генерується безмежно(рис. 1.8), це випливає в те, що гравець використовує лише незначну частину згенерованого безмежного світу під час гри, а інша частина лише займає пам’ять.

1.3 Постановка задач проектування до розроблювальної системи

Аналіз предметної області показав, які особливості процедурної генерації потрібно виділити зважаючи на переваги та недоліки розглянутих рішень.

Особливості процедурної генерації:

- Рівень має складатись із кімнат та переходів між ними(коридорів);
- Кожна кімната має бути з’єднана хоча б з одною іншою кімнатою за допомогою коридорів;
- Кожна кімната має бути з’єнана зі всіма іншими кімнатами безпосередньо, або через інші кімнати;
- Алгоритм має генерувати кімнати різного розміру, та має підпорядковуватись встановленному середньому значенню розміра кімнат;
- Алгоритм має генерувати рівень різних розмірів в залежності від вподобань гравця;
- Алгоритм має підлаштовувати «лінійність» рівня в залежності від вподобань гравця;
- Алгорим має генерувати певну кількість ворогів та їх рівень в залежності від розмірів певної кімнати, вибранної складності гри та підлаштовуючись під вподобання гравця;
- Алгоритм має генерувати систему освітлення кімнати та коридорів в залежності від їх розмірів;

- Алгоритм має випадково обирати один з видів інтер’єру кімнати, в залежності від розмірів кімнати;
- Рівань має містити від однієї до трьох «прихованих» кімнат, різних видів складності;
- Алгоритм має генерувати «приховані» кімнати в залежності від вподобань гравця;
- Алгоритм має записувати дії гравця при проходженні рівня, аналізувати дані та підналаштовувати вподобання гравця відповідно до його дій під час проходження;
- Алгоритм має підлаштовуючись під вподобання гравця генерувати рівні найбільш підходящі гравцю.

Отже, при кожному проходжені гравцем процедурно генеруємого рівня, його дії та результати проходження будуть записуватися програмно, оброблятися та відповідно отриманих даних будуть підналаштовуватися вподобання гравця. При генерації нового рівня ці вподобання будуть отримуватись і в залежності від них буде налаштовуватись різні аспекти генеруємого рівня.

Висновки до розділу

1. Були розглянуті існуючи рішення.
2. Визначені основні переваги та недоліки розглянутих рішень.
3. Проаналізовані основні особливості типових систем.
4. Сформовано основні вимоги для розроблюваного модулю.

На базі аналізу розглянутих алгоритмів процедурної генерації були визначені такі основні компоненти: рівень має мати один основний шлях проходження, рівень має містити приховані зони, рівень має бути легко маштабуємий, алгоритм має підлаштовуватись під конкретного гравця, алгоритм має генерувати рівень різного маштабу та складності.

Одні з основних недоліків оглянутих алгоримів це: надлишковість та перенавантаженість, залежність від неявних систем розподілу, не завжди достатня свобода вибору шляхів проходження генеруємих рівнів.

Після аналізу існуючих алгоритмів процедурної генерації були описані загальні вимоги до проектуємого алгоритму.

РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ ТА ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Вибір архітектури та платформи розробки

2.1.1 Вибі ігрового движка

Unity - це міжплатформне середовище розробки комп'ютерних ігор, що дозволяє створювати ігри під більшість популярних платформ. За допомогою даного середовища розробляються ігри, які запускаються на персональних комп'ютерах (які працюють під Windows, MacOS, Linux), на смартфонах і планшетах (iOS, Android, Windows Phone), на ігрових консолях (PS, Xbox, Wii) та інших[6].

На Unity написані тисячі ігор, додатків та симуляцій, які охоплюють безліч платформ і жанрів. При цьому Unity використовується як великими розробниками, так і незалежними студіями.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Движок підтримує дві скриптові мови: C #, JavaScript (модифікація. Розрахунки фізики виробляє фізичний движок PhysX від NVIDIA.

Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у об'єктів є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого об'єкта на сцені обов'язково присутній компонент Transform - він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою.

Також Unity підтримує фізику твердих тіл і тканини, а також фізику типу Ragdoll (тряпічная лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

Unity 3D підтримує систему Level Of Detail (скор. LOD), суть якої полягає в тому, що на далекій відстані від гравця високодеталізовані моделі замінюються на менш деталізовані, і навпаки, а також систему Occlusion culling, суть якої в тому, що у об'єктів, що не потрапляють в поле зору камери не візуалізується геометрія і колізія, що знижує навантаження на центральний процесор і дозволяє оптимізувати проект. При компіляції проекту створюється виконуваний (.exe) файл гри (для Windows), а в окремій папці — дані гри (включаючи всі ігрові рівні і спільні бібліотеки).

Движок підтримує безліч популярних форматів. Моделі, звуки, текстури, матеріали, скрипти можна запаковувати в формат .unityassets і передавати іншим розробникам, або викладати у вільний доступ. Цей же формат використовується у внутрішньому магазині Unity Asset Store, в якому розробники можуть безкоштовно і за гроші викладати в загальний доступ різні елементи, потрібні при створенні ігор. Щоб використовувати Unity Asset Store, необхідно мати акаунт розробника Unity. Unity має всі потрібні компоненти для створення мультиплеєра. Також можна використовувати відповідний користувачеві спосіб контролю версій. Наприклад, Tortoise SVN або Source Gear.

Використання програмного засобу Unity 2018 зумовлено тим, шо він:

- Використовує мову C#. Дана мова високорівнева і дозволяє розробнику легко перейти до розробки гри. Це важливий момент, тому що на відміну від інших засобів, де використовується мова C ++, в C # є багато елементів і прийомів, які вже реалізовані, і програмісту необхідно тільки скористатися ними.

- Є кроссплатформним, тобто один і той же код, написаний на Unity, з мінімальними змінами може бути перенесений на різні платформи (PC, Mac, Android, iOS, Web, ігрові консолі). Це величезний плюс, який скорочує час на розробку гри в кілька разів.
- Має гарне Community. Це означає, що у різних функцій Unity є чіткий опис з прикладами на сайті розробника, звернутися до якого можна у будь-який момент. Якщо щось все ж залишилося незрозумілим, служба підтримки обов'язково відповість на виниклі питання.
- Має сервіс Asset Store, де є величезна кількість різних плагінів і ресурсів для створення гри. Зрозуміло, якісь з них безкоштовні, якісь платні, але всі вони зібрані в одному місці зі зручним пошуком і можливістю завантажити, інтегрувати і отримати відразу робочий функціонал.
- Сучасний рівень графіки, здатний конкурувати з іншими движками. Unity, безумовно, програє UnrealEngine за кількістю реалізованих можливостей. Однак Unity володіє такими можливостями, як deferred освітлення, стандартний набір постпроцессінгових ефектів, SSAO, прискорена опрацювання лайтмапов.
- Гідним чином опрацьований фізичний движок.
- Масштабованість і продуктивність. Більшу частину простих процесів движок обробляє на чудовому рівні.
- Запуск будь-якої програми на Unity в веб-плагін.

Як альтернативу для міжплатформного середовища розробки комп'ютерних ігор Unity можна розглядати ігровий движок Unreal Engine.

Unreal Engine написаний на мові C++, движок дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X; консолей Xbox, Xbox 360, Xbox One,

PlayStation 2, PlayStation 3, PlayStation 4, PSP, PS Vita, Wii, Dreamcast, GameCube і ін, а також на різних портативних пристроях, наприклад, пристрої Apple (iPad, iPhone), керованих системою iOS та інших. (Вперше робота з iOS була представлена в 2009 році, в 2010 році продемонстрована робота движка на пристрої з системою webOS)[7].

Для спрощення портування движок використовує модульну систему залежних компонентів; підтримує різні системи візуалізації (Direct3D, OpenGL, Pixomatic; у ранніх версіях: Glide, S3, PowerVR), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше: A3D), засоби голосового відтворення тексту, розпізнавання мови, модулі для роботи з мережею і підтримки різних пристроїв введення.

Для гри по мережі підтримуються технології Windows Live, Xbox Live, GameSpy та інші, включаючи до 64 гравців (клієнтів) одночасно. Таким чином, движок адаптували і для застосування в іграх жанру MMORPG.

Однією з основних особливостей, запланованих для UE4, було глобальне освітлення в режимі реального часу з використанням трасування воксельных (2д пікселі у тривимірному просторі) конусів, що виключає попередньо обчислена освітлення. Однак ця функція була замінена аналогічною, але менш дорогим алгоритмом до випуску для всіх платформ, включаючи ПК, через проблеми з продуктивністю. UE4 також включає нові функції розробника для скорочення часу ітерації і дозволяє оновлювати код C++ під час роботи ядра. Нова система візуальних сценаріїв" Blueprint "(наступник Ue3 "Kismet") дозволяє швидко розробляти ігрову логіку без використання C++, і включає в себе живе налагодження. У результаті скорочується час ітерації і зменшується розрив між технічними художниками, дизайнерами і програмістами.

UnrealScript (часто скорочено UScript) є рідним скриптовою мовою Unreal Engine, використовуваним для створення ігрового коду та ігрових подій до випуску Unreal Engine 4. Мова був розроблений для простого,

високорівневого програмування ігор. Інтерпретатор UnrealScript був запрограмований Тімом Суїні, який також створив більш ранній мова сценаріїв гри.

Подібно Java, UnrealScript є об'єктно-орієнтованим без множинного спадкування (всі класи успадковуються від загального класу об'єктів), а класи визначаються в окремих файлах з ім'ям класу, який вони визначають. На відміну від Java, UnrealScript не має об'єктних оболонок для примітивних типів. Інтерфейси підтримуються тільки в третьому поколінні Unreal Engine і декількох іграх Unreal Engine другого покоління. UnrealScript підтримує перевантаження операторів, але не перевантаження методів, за винятком необов'язкових параметрів.

На конференції розробників ігор 2012 року Epic оголосила, що UnrealScript був видалений з Unreal Engine 4 на користь C++. Візуальні сценарії будуть підтримуватися системою візуальних сценаріїв Blueprints, заміною більш ранньої системи візуальних сценаріїв Kismet.

Переваги:

- Unreal engine в цілому побудований на перевіреній структурі багатьох AAA ігор. Це високорозвинений двигун якому вже більше десяти років.
- Він має чудовий пайплайн. Створення шейдерів на основі вузлів завжди було величезною перевагою для UE4 протягом тривалого часу.
- Повністю відкритий вихідний код. Спільнота Epic постійно допомагає покращувати движок. Ви можете побачити список учасників Non-epic в кожному великому випуску. Крім того, ви можете копатися в джерелі, щоб знайти, як деякі функції працюють на більш низькому рівні. Якщо ви будуєте з c++ , ви можете побудувати з джерела , чи ні. Якщо ви будуєте з допомогою blueprint, ви можете знайти, як деякі функції працюють на більш низькому рівні.

- Blueprint (візуальне Програмування)-це потужний інструмент для всього движка, який може бути легко розширений, якщо у вас є Додаткова робоча сила в C++.
- Тонни навчального матеріалу

Недоліки:

- Дуже примітивний магазин додатків.
- Високий поріг входження.
- Потреба великої кількості ресурсів.

2.1.2 Вибір мови програмування

C # (вимовляється як "сі Шарп") — проста, сучасна об'єктно-орієнтована мова програмування. C # відноситься до широко відомого сімейству мов C, і здасться добре знайомим кожному, хто працював з C, C ++, Java або JavaScript[8].

Синтаксис C # дуже багатий, але при цьому простий і зручний у вивченні. Характерні фігурні дужки C # миттєво впізнаються всіма, хто знайомий з C,
C ++ або Java. Розробники, які знають будь-яку з цих мов, зазвичай дуже швидко починають ефективно працювати в C #. Синтаксис C # спрощує багато складності C ++, але при цьому надає відсутні в Java потужні функції, наприклад обнуляє типи значень, перерахування, делегати, лямбда-вирази і прямий доступ до пам'яті. C # підтримує універсальні методи і типи, які забезпечують більш високий рівень безпеки і продуктивності, а також ітератори, що дозволяють визначати в класах колекцій власну поведінку ітерації, яку можна легко застосувати в клієнтському коді. Вирази LINQ створюють дуже зручну мовну конструкці.

C # є об'єктно-орієнтованою мовою, а значить підтримує інкапсуляцію, успадкування і поліморфізм. Всі змінні і методи, включаючи

метод Main, який представляє собою точку входу в додаток, ынкапсулюються в визначення класів. Клас успадковується безпосередньо з одного батьківського класу, але може реалізовувати будь-яке число інтерфейсів. Методи, які скасовують віртуальні методи батьківського класу, повинні містити ключове слово override, щоб виключити випадковий перевизначення. У мові C # структура схожа на полегшений клас: це тип, що розподіляється в стеці, який реалізує інтерфейси, але не підтримує спадкування[8].

Крім цих основних принципів об'єктно-орієнтованого програмування, C# пропонує ряд інноваційних мовних конструкцій, що спрощують розробку програмних компонентів:

- Інкапсульовані сигнатури методів, іменовані делегатами, які дозволяють реалізувати повідомлення про події.
- Властивості, що виконують функцію акцессоров для закритих змінних-членів.
- Атрибути, що надають декларативні метадані про типи під час виконання.
- Внутрістрочние коментарі для XML-документації.
- LINQ для створення запитів до різних джерел даних.

Для взаємодії з іншим програмним забезпеченням Windows, наприклад з об'єктом COM або власними бібліотеками DLL Win32, ви можете застосувати процес C #, відомий як "Взаємодія". Взаємодія дозволяє програмам на C # робити практично все, що можливо в додатку машинного коду C ++. C # підтримує навіть покажчики і поняття "небезпечного" коду для тих випадків, в яких критично важливий прямий доступ до пам'яті.

Процес побудови в C # простіше в порівнянні з C або C ++, але більш гнучкий, ніж в Java. Окремі файли заголовка не використовуються, і немає необхідності оголошувати методи і типи в певному порядку.

Вихідний файл C # може визначити будь-яке число класів, структур, інтерфейсів і подій.

Переваги мови C#:

- Мова програмування C # претендує на справжню об'єктну орієнтованість (будь-яка мовна сутність претендує на те, щоб бути об'єктом);
- Підтримуються різні стилі і технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси);
- Компонентно-орієнтований підхід до програмування, що сприяє меншій машинно-архітектурної залежності результуючого програмного коду, гнучкості, переносимості і легкості повторного використання (фрагментів) програм;
- Орієнтація на безпеку коду (в порівнянні з C і C ++);
- Уніфікована система типізації;
- Розширена підтримка подієво-орієнтованого програмування.

Мовою - аналогом C# в Unity може бути JavaScript.

JavaScript — мультипарадигмна мова програмування. Підтримує об'єктно-орієнтований, імперативний і функціональний стилі. Є реалізацією мови ECMAScript[9].

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додавання інтерактивності веб-сторінок.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожою на Java, але при цьому легкою для використання непрограмістів. Мовою JavaScript не володіє будь-яка компанія або

організація, що відрізняє її від ряду мов програмування, використовуваних в веб-розробці

JavaScript є об'єктно-орієнтованою мовою, але прототипування що використовується в мові обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними клас-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, анонімні функції, замикання - що додає мові додаткову гнучкість.

Мова також не позбавлена недоліків. Найбільш значимі з них:

- Мова компілюється в момент виконання коду. Кожен раз, коли ви відкриваєте сайт, javascript код починає компілюватиметься. Як мінімум збільшується час виконання програми.
- Відсутня типізація даних. Проблема всіх скриптових мов. Поки виконання коду не дійде до потрібного рядка, не зрозуміло чи вона. А значну частину з пошуку помилок міг би взяти на себе компілятор, якби знав типи даних, з якими він працює. Та й по швидкості виконання, типізований код швидше.
- Незвична для багатьох програмістів об'єктна модель. Класи і наслідування класів присутні, але вони сильно відрізняються від звичної багатьом реалізацій в мовах програмування C ++ / C # / Java.

Незважаючи на схожий з Сі синтаксис, JavaScript в порівнянні з мовою Сі має корінні відмінності:

- об'єкти з можливістю інтроспекції,
- функції як об'єкти першого класу,
- автоматичне приведення типів,
- автоматичне прибирання сміття,
- анонімні функції.

У мові відсутні такі корисні речі, як:

- Стандартна бібліотека: зокрема, відсутній інтерфейс програмування додатків по роботі з файловою системою, управління потоками введення-виведення, базових типів для бінарних даних.
- Стандартні інтерфейси до веб-серверів і баз даних.
- Система управління пакетами, яка б відстежувала залежності і автоматично встановлювала їх.

2.1.3 Вибір програмного середовища

Microsoft Visual Studio — це програмне середовище для розробки додатків для ОС Windows, як консольних, так і з графічним інтерфейсом[10].

У комплект входять наступні основні компоненти:

- Редактор вихідного коду, який включає безліч додаткових функцій, як автодоповнення IntelliSense, рефракторинг коду і т. д .;
- Відладчик коду;
- Редактор форм, призначений для спрощеного конструювання графічних інтерфейсів;
- Веб-редактор;
- Дизайнер класів;
- Дизайнер схем баз даних.

Visual Studio також дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додава ння підтримки систем контролю версій вихідного коду (Subversion і VisualSourceSafe), додавання нових наборів інструментів (для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів процесу розробки програмного забезпечення)[10].

Інтегроване середовище розробки (IntegratedDevelopmentEnvironment — IDE) Visual Studio пропонує ряд

високорівневих функціональних можливостей, які виходять за рамки базового управління кодом.

Нижче перераховані основні переваги IDE-середовища Visual Studio:

- Підтримка безлічі мов при розробці.
- Менше коду для написання. Для створення більшості додатків потрібна невелика кількість стандартного стереотипного коду.
- Інтуїтивний стиль кодування.
- Більш висока швидкість розробки.
- Можливості налагодження.
- Можливість управління проектом.
- Вбудована функція управління вихідним кодом.
- Можливість рефакторизації коду.

Аналогом Visual Studio як середовища розробки для мови C# може бути Sharp Develop.

SharpDevelop – безкоштовне середовище розробки для C #, Visual Basic .NET, Boo, IronPython, IronRuby, F #, C ++. Зазвичай використовується як альтернатива Visual Studio .NET[11].

SharpDevelop 2.0 надає інтегрований відладчик, який використовує власні бібліотеки і взаємодіє з виконуючим середовищем .NET через COM Interop.

Можливості та особливості

- Написане повністю на C #.
- Підсвічування синтаксису для C #, IronPython, HTML, ASP, ASP.NET, VBScript, VB.NET, XML, XAML.
- Візуальний редактор для WPF і форм Windows Forms (COM-компоненти не підтримуються).
- Інтегрована підтримка NUnit, MbUnit і NCover.
- Інтегрована підтримка аналізатора збірок FxCop.
- Інтегрований відладчик.

- Інтегрований профайлер.
- Інтегрована підтримка SVN, Mercurial і Git.
- Конвертор коду між мовами C #, VB.NET, IronPython і Boo.
- Перегляд документації, отриманої з документуючих коментарів.
- Можливість розширення зовнішніми інструментами.
- Можливість розширення на основі механізму Add-Ins.

Переваги:

- Середовище безкоштовне.
- Зовні дуже схоже на Microsoft Visual Studio.
- Може працювати з проектами Visual Studio, яка, в свою чергу, може працювати з проектами SharpDevelop'a.
- У SharpDevelop вбудована утиліта NDoc призначена для формування документації по коду на основі xml коментарів в самому коді. На жаль робота над проектом NDoc закінчилася ще в 2006 році, так що утиліта на даний момент застаріла, однак цілком функціональна.

Недоліки:

- На перший погляд проста і гнучка система підсвічування вихідного коду. Однак, щоб змусити підсвітити, наприклад, будь-яким кольором тип поля в класі або тип значення будь-якого методу (як в MS VS) вимагає від користувача складного налаштування;
- Неможливо подивитися значення будь-якої змінної в debug-режимі простим наведенням курсора мишки на неї;
- Відсутність watch листа, де б ці змінні можна було подивитися. Є певна подоба watch листа, але подивитися там можна далеко не все.

2.2 Проектування структури підмодулю процедурної генерації

Для спрощення проектування і розробки алгоритму було вирішено використати систему розбивки основної задачі на окремі підзадачі. Основна задача є створення модулю процедурної генерації з елементами штучного інтелекту. Було вирішено розбити основну задачу на дві підзадачі, на два

підмодулі, перший буде представляти собою підмодуль визначення вподобань гравця, що буде включати в собі елементи штучного інтелекту, інший підмодуль є самою процедурною генерацією рівня.

Підмодуль генерації рівня буде скриптом для міжплатформного середовища розробки комп'ютерних ігор Unity та написаний на мові C#. Він буде відповідати за всю генерацію рівня і представляти зручні інструменти розробки для процедурної генерації. Даний підмодуль буде давати змогу не лише згенерувати рівень з стандартними значеннями, а також розробник, використовуючи цей підмодуль, може повністю налаштувати бажані значення для генерації, такі як, наприклад, розмір рівня, його форма, кількість кімнат, середній розмір кімнат та максимальне значення різниці між найбільшою згенерованою кімнатою та її середнім розміром, кількість коридорів, їх розміри.

Алгоритми процедурної генерації в розролювальній системі можуть мати два різних сценарії їх використання.

Перший сценарій, це те, що алгоритми будуть використані розробником, що відповідає за розробку рівнів гри, для стоврення базової конструкції розроблюваного рівня, що будуть відповідати конкретним вимогам які стоять перед розробником. Після констрюювання рівня, він може бути переданий гейм дизайнеру, що буде працювати над стилістикою та основними елементами інтер’єру. Після цього кроку розробка рівня буде завершена.

Також, для використання підмодулю, гейм дизайнеру потрібно буде задати матеріали з яких будуть генеруватися стіни, підлога та стеля. Крім цього потрібно буде помістити у відповідну папку з ресурсами заготовлені заздалегідь об’єкти інтер’єру та єлементи освітлення, які будуть використані алгоритмом для генерації елментів декору створюваних кімнат.

Після того як будуть виконані всі підготовчі дії підмодуль може

бути використаний для процедурної генерації рівня, із значеннями за замовчуванням, встановленими гейм дизайнером, або встановленими підмодулем, що відповідає за вподобання конкретного гравця.

2.3 Проектування підмодулю з елементами штучного інтелекту

Іншим сценарієм використання алгоритмів передбачається, що модуль процедурної генерації буде використовуватись як самодостатній та автономний елемент генерації готового рівня під час гри. В данному випадку алгоритми процедурної генерації мають створювати повністю завершений рівень, який не потребує втручань нікого зовні.

Саме за таким сценарієм буде працювати підмодуль з елементами штучного інтелекту, що визначає вподобання гравця. Підмодуль буде розділятись на два фрагменти.

Перший фрагмент має записувати і зберігати реазультати проходження гравцем вже згенерованого рівня, має приділяти особливу увагу таким моментам як, час проходження гравцем рівня, середня кількість кроків затрачених на здолання ворогів, матрицю відкритих гравцем ділянок, кількість знайдених «прихованих» кімнат, кількість поборених ворогів.

Інший фрагмент модулю на основі збережених данних проходження гравцем попередніх згенерованих рівнів буде визначати вподобання конкретного гравця, та його стиль проходження рівня. Ці вподобання будуть зберігатися та змінюватись по мірі проходження гравцем гри. Чим більше буде зібрано матеріалу першим фрагментом підмодулю, чим більше разів гравець буде проходити згенеровані рівні, тим точніше будуть визначені вподобання данного гравця і відповідно згенеровані рівні більш відповідні до стилю гри конеретного гравця.

2.4 Взаємодія підмодулів

Після проектування обох підмодулів була створена схема їх взаємдодії один між одним.

Після того як підмодуль, що відповдіє за визначення вподобань конкретного гравця набере достатньо данних для формування стилю гри конкретного гравця (достатня кількість данних має набиратись вже після проходження першого процедурно згенерованого рівня), підмодуль буде налаштовувати алгоритми процедурної генерації відповідно до вподобань гравця.

Процедура налаштування параметрів алгоритму буде викликатись кожен раз, коли підмодуль буде отримувати нову порцію данних. Таким чином, з достатньою кількістю ітерації алгоритм процедурної генерації повністю буде відповідати вподобанням гравця до проходження гри.

Висновки до розділу

1) Вибрано архітектуру та платформу, на якій буде здійснюватися розробка.

2) Опис архітектури та допоміжних засобів, які потрібні для розробки системи.

3) Спроектовано підмодуль процедурної генерації.

4) Спроектовано підмодуль з елементами штучного інтелекту.

5) Спроектовано взаємодію підмодулів.

Отже, для розробки було обрано міжплатформне середовище розробки Unity. Основним аспектом вибору стали потужний фізичний двигун, сучасна графіка та кросплатформенність. Для реалізації модулю цього, також, обрано мову програмування C#. Та у якості середовища розробки було обрано Visual Studio.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СПРОЕКТОВАНОГО МОДУЛЮ ТА ТЕСТУВАННЯ СИСТЕМИ.

3.1 Реалізація підмодулю процедурної генерації

В попередньому розділі було спроектовано модуль процедурної генерації та його підмодулі, тож тепер можемо приступити до їх розробки.

Для зручності розробки та проектування підмодуль процедурної генерації було вирішино розбити на дві підсистеми:

- Підсистема генерації кімнат та коридорів;
- Підсистема розміщення об’єктів, генерації освітлення та інтер’єру;

Також, для збереження данних та спрощення розробки підмодулю, було вирішено додати декілька допоміжних класів, всі вони будуть розглянуті перед ознайомленням з основними підсистемами, для більшого розуміння розроблених алгоритмів.

3.1.1 Допоміжні класи

Клас `Point`

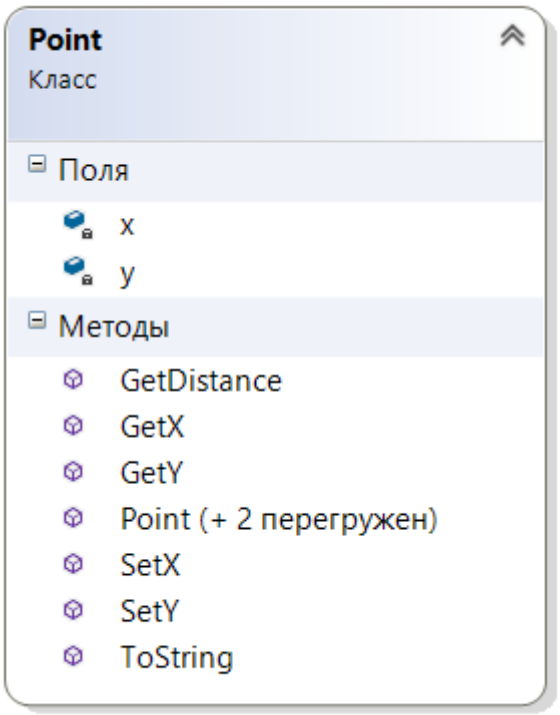


Рисунок 3.1 Диаграмма класу `Point`

Клас `Point` (рис. 3.1) представляє собою допоміжний клас, призначення якого полягає в тому щоб зберігати позицію по осі X та Y точки об’єкту, позиції зберігаються в полях класу як змінні типу «int»;

Також клас реалізує декілька методів для роботи з цими полями. Клас має три перевантажені конструктори, які приймають різні типи данних: «пустий» конструктор, конструктор що приймає тип Point та конструктор що приймає тип «int».

Публічні методи `GetX, SetX, GetY, SetY` використовуються для доступу та модифікації полів «X» та «Y» відповідно.

Публічний метод `GetDistance` приймає в якості аргумента інший єкземпляр класу `Point`, та дозволяє отримати відстань між цими двома точками.

Клас `PointPair`

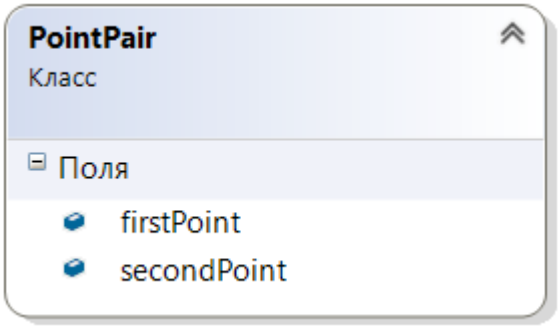


Рисунок 3.2 Диаграмма класу `PointPair`

Клас `PointPair` (рис 3.2) представляє собою простий проміжний клас, призначений для збереження двох єкземплярів класу `Point`.

Клас Map

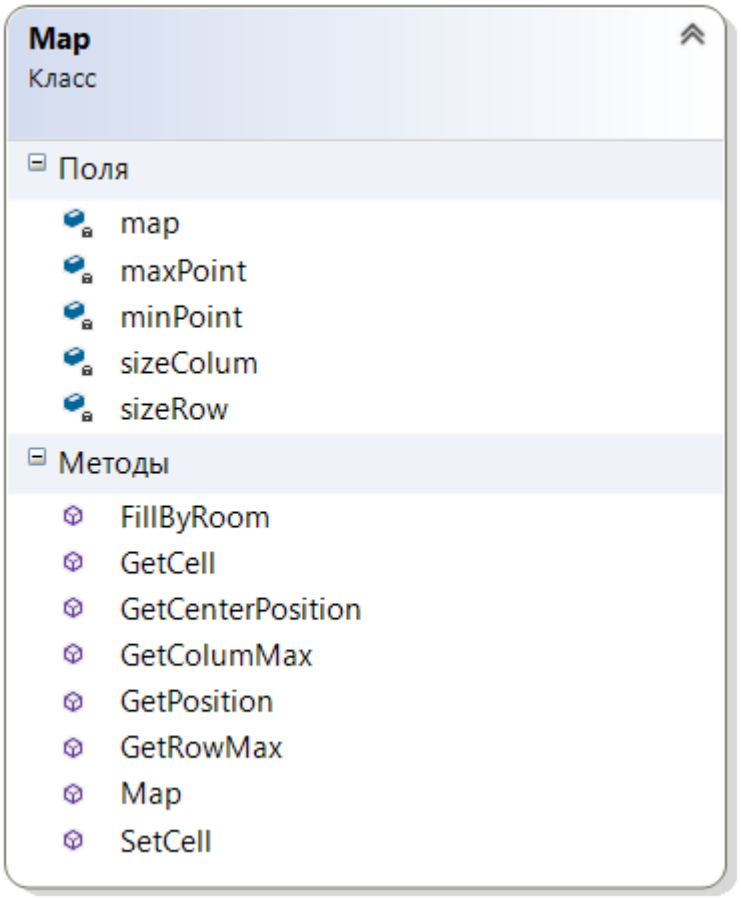


Рисунок 3.3 Диаграмма класу Map

Клас Map (рис 3.3) представляє собою допоміжний клас, призначенний для зберігання матриці структури рівня, першої та останної комірки рівня, та його розміри в комірках.

Весь рівень під час генерації розбивається на комірки розміром, який був заданий в параметрах процедурної генерації, та записується в матрицю данного класу. Поля MaxPoint, MinPoint є єкземплярами класу Point та зберігають крайню верхню ліву точку та крайню праву нижню точку рівня відповідно.

Конструктор класу не має перевантажень, та приймає чотири аргументи, розмір рівня в комірках, верхню ліву точку та праву нижню точку, та записує їх значення в поля sizeColum, sizeRow, maxPoint, minPoint відповідно.

Методи `GetCell, GetColumMax, GetRowMax` дозволяють отримати доступ до полів класу, `map, sizeColum, sizeRow` відповідно.

Методи `GetPosition` та `GetCenterPosition` викликаються для отримання координат комірки, її верхньої лівої точки та центру відповідно.

Метод FillByRoom приймає в якості аргументу єкземпляр класу Room, та заповює матрицю структури відповідно до нього.

3.1.2 Підсистема генерації кімнат та коридорів

Клас `Room`

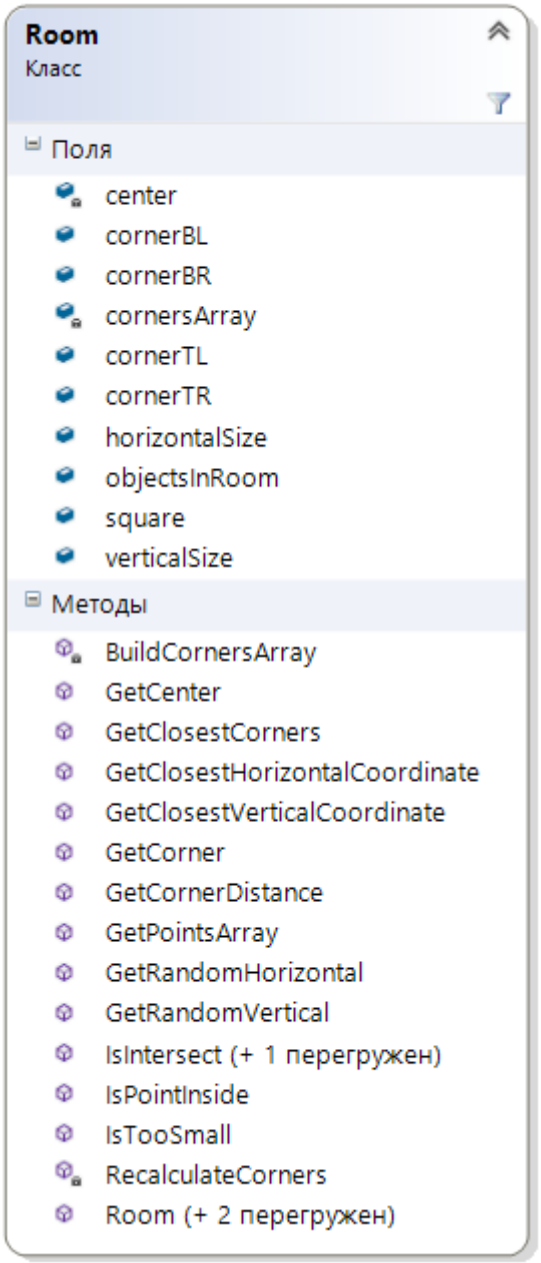


Рисунок 3.4 Диаграмма класу Room

Клас Room (рис. 3.4) призначенний для генерації та збереження інформації про кімнату, в ньому реалізовані всі основні методи для реалізації генерації кімнати та генерації коридорів між кімнатами.

Клас Room зберігає основну інформацію про кімнату, а саме:

- center — єкземпляр класу Point, який відповідає точці центру кімнати
- corner TL, TR, BL, BR — єкземпляри класу Point, відповідають координатам верхнього лівого, верхнього правого, нижнього лівого, нижнього правого куту кімнати, відповідно.
- vertical-horizontal size — відповідають ширині та довжині кімнати в комірках.
- square — площа генеруємої кімнати.
- objectsInRoom — масив єлементів які знаходяться в кімнаті, включає в себе як єлементи інтер'єру так і моделі ворогів.

Конструктор данного класу має три реалізації, «пустий» коструктор, конструктор, який приймає у якості аргументів чотири параметри : координати центру і розміри кімнати та конструктор, що приймає два єкземпляри класу Point, що відповідають координатам лівого верхнього та правого нижнього куту.

Метод BuildCornerArray генерує массив з координатів кутів, для поточного єкземпляру класу.

Основний алгоритм процедурної генерації кімнати:

```
Room GenerateRandomRoom()
    {
        float xRandom = Random.Range(-1f, 1f);
        float yRandom = Random.Range(-1f, 1f);
```

```
            int cx = (int)(Sign(xRandom) * dSize *
FilterRandom(true, Mathf.Abs(xRandom)) / (2 *
Mathf.Pow(propCoefficient, Mathf.Abs(whProp))));
            int cy = (int)(Sign(yRandom) * dSize *
FilterRandom(false, Mathf.Abs(yRandom)) / (2 *
Mathf.Pow(propCoefficient, Mathf.Abs(whProp))));
            int w = Random.Range(roomSize - roomSizeDelta,
roomSize + roomSizeDelta);
            int h = Random.Range(roomSize - roomSizeDelta,
roomSize + roomSizeDelta);
            return new Room(cx, cy, w, h);
        }
    float FilterRandom(bool isWidth, float value)
            if (isWidth)
                return value * Mathf.Pow(propCoefficient,
whProp);
            else
                return value * Mathf.Pow(propCoefficient, -1
* whProp);
```

Контроллер виконує метод `GenerateRandomRoom`, який відповідно до заданих, гейм дизайнером, або підмодулем штучного інтелекту, значень параметрів процедурної генерації `dSize` (бажаний розмір рівня), `propCoefficient` (бажане співвідношення довжини рівня, до його ширини), отримує псевдивипадкове значення координат центру кімнати та її довжини та ширини.

Використовуючи параметри отримані в результаті виконання методу `GenerateRandomRoom` викликається конструктор класу `Room`, у який в якості аргументів передаються отримані параметри.

Після створення нового єкземпляру класу виконується перевірка чи кімната не переається з іншими кімнатами, та чи її розміри відповідають параметрам за допомогою описаних нище алгоритмів:

```
    if (!newRoom.IsTooSmall(coridorThicknes) &&
!newRoom.IsIntersect(rooms))
    public bool IsIntersect(Room room)
        {
            //перевіряєм, чи знаходяться кути першої кімнати
в другій, а потім кути другої кімнати в першій. Якщо хоча б
один раз - так, то кімнати перетинаються
            if (IsPointInside(room.GetCorner(0)) ||
IsPointInside(room.GetCorner(1)) ||
IsPointInside(room.GetCorner(2)) ||
IsPointInside(room.GetCorner(3)) ||
                room.IsPointInside(cornerBL) ||
room.IsPointInside(cornerTL) || room.IsPointInside(cornerTR)
|| room.IsPointInside(cornerBR))
            {
                return true;
            }
            //далі перевіряємо чи накладається одна кімната
на другу. Дивимось чи центр другої кімнати лежить на
горизонталі або вертикалі першої
            Point c = room.GetCenter();
            if ((c.GetX() >= cornerBL.GetX() && c.GetX() <=
cornerBR.GetX() && center.GetY() >= room.GetCorner(0).GetY()
&& center.GetY() <= room.GetCorner(1).GetY())
                || (c.GetY() >= cornerBL.GetY() && c.GetY()
<= cornerTL.GetY() && center.GetX() >=
```

```
room.GetCorner(0).GetX() && center.GetX() <=
room.GetCorner(3).GetX()))
            {
                return true;
            }
            return false;
        }
```

Алгоритм перевірки під час виконання викликає два методи класу Room, а саме методи `IsPointInside` та `GetCorner`, перший метод приймає у якості аргументу єкземпляр класу `Point` та повертає бульове значення, чи знаходяться координати даної точка у межах перевіряємої кімнати, чи ні. Метод `GetCorner` повертає єкзепляр класу `Point`, що відповідає запитованому куту кімнати.

Після проходження перевірки, в залежності від результатів, кімната, або створюється заново викликаючи метод `GenerateRandomRoom`, або зберігається в массиві вірно згенерованих кімнат.

Далі алгоритм переходить до створення коридорів між кімнатами, виконується перевірка, чи масив кімнат містить хоча б одну кінату, якщо так, то для кожної виконується пошук найближчої сусідньої кімнати використовуючи метод `GetRoomToRoomClosestIndex`, код якого приведений нижче.

```
    int GetRoomToRoomClosestIndex(int i)
        {
            float closestDistance = 2 * dSize;
            Room room = rooms[i];
            int closestIndex = -1;
            for (int j = 0; j < rooms.Count; j++)
            {
                if (i != j)
                {
                    //вираховуємо відстань між кожною
вершиною кімнат, та обираємо мінімальну
```

```
                    float d =
room.GetCornerDistance(rooms[j]);
                    if (d < closestDistance)
                    {
                        closestDistance = d;
                        closestIndex = j;
                    }
                }
            }
            return closestIndex;
```

Використовуючи метод `GetRoomToRoomClosestIndex` алгоритм для кожної кімнати вираховує масив найближчих сусдніх кінат розташованих у порядку збільшення відстані між ними.

З цього масиву береться лише декілька елементів, кількість яких залежить від значення параметру `CoridorCount`. Також якщо значення цього параметру перевищує кількість сусідніх кімнат, то будуються коридори до всіх.

Для побудування коридорів використовується метод `BuildCoridor` який приймає у якості аргументів індекси кімнат між якими і будується коридор.

Алгоритм методу `BuildCoridor` наступний:

- У стартовій кімнаті обираються дві точки за допомогою методів `GetRandomVertical` та `GetRandomHorizontal`, функція яких полягає в тому, що випадковим чином обираються точки по горизонталі горизонталі та вертикалі приміщення, враховуючи ширину коридору який буде генеруватись. Точки не повинні бути занадто близько до краю кімнати, що б призвело до того, що коридор згенерувався за межами кімнати. Знайдені точки відповідають координаті на одній із стінок кімнати, де буде починатись коридор.
- Наступним кроком алгоритм знахожить найближчи точку до координати початку коридору, на одній із стінок, іншої кімнати. Для цього використовуються методи `GetClosestVerticalCoordinate` та `GetClosestHorizontalCoordinate` класу `Room`, призначення

яких полягає в знаходженні найближчої точки, на одній із стінок кімнати, відповідно до отриманих координат таким чином, що пряма проведена з точки початку коридору однієї кімнати, обов’язково перетиналася з прямою проведеною з іншої кімнати.

- Після того як координати були знайдені для обох приміщень, будується два прямих коридори, які починаються в відповідних обраних точках кімнати, мають заздалегіть зазначену параметрами ширину, та мають довжину, яка співпадає з відстанню на якій прямі проведені з кімнат перетнуться.
- Останнім кроком коридор додається до масиву коридорів, на цьому алгоритм генерації коридору між кімнатами вважається завершеним.

Після того, як структура кімнат та коридорів була успішно сформована, формується матриця прохідності, вона являє масив, що представляє собою схематичне зображення згенерованого рівня, що містить у собі всі кімнати та переходи між ними.

Ця матриця є екземпляром классу `Map`, який зберігає всі точки та ставить їх у відповідність до сформованих кімнат та коридорів.

Наступним кроком в алгоритмі процедурної генерації після формування структури рівня, є його побудова, для цього використовується метод `EmitGeometry` він приймає у якості аргументів об’єкти що відповідають стінкам заданим для генерування конкретного рівня.

Функція методу `EmitGeometry` полягає в тому, щоб, використовуючи згенеровану матрицю прохідності, та дані об'єкти (частини стінок та підлоги та стелі), побудувати рівень відповідно до структури, яка зберігається в матриці прохідності.

Для цього алгоритм проходить по кожному елементу матриці, з'ясовує відповідає дана точка, елементам кімнати, коридору, або є пустою, з'ясовується позиція точки в кімнаті. Якщо дана точка знаходиться в межах кімнати або коридору формується підлога та стеля, або ж якщо точка

знаходиться на краю кімнати або коридору, окрім підлоги та стелі, формується пряма стінка або вугол з елементів які були обрані гейм дизайнером.

Після завершення методу `EmitGeometry` був згенерований рівень, що містить стіни, підлогу та стелю.

На цьому підсистема генерації кімнат та коридорів завершує свою роботу та передає контроль підсистемі розміщення об’єктів, генерації освітлення та інтер’єру.

3.1.3 Підсистема розміщення об’єктів, генерації освітлення та інтер’єру.

Після того як підсистема генерації кімнат та коридорів побудувала пусті приміщення, вони мають бути заповнені елементами інтер'єру, освітлення, об'єктами для взаємодії. Цим все може бути виконано окремо людиною яка відповідає за дизайн рівнів, або може бути згенеровано автоматично алгоритмом процедурної генерації.

Підсистема автоматичної генерації включає в себе такі функції як:

- Генерація освітлення в кімнатах;
- Генерація освітлення в коридорах;
- Генерація об'єктів інтер'єру в кімнатах;
- Генерація ворогів в межах кімнати.

Алгоритм генерації освітлення в кімнатах виклакає метод `AddLampToTheRoom` для кожної кімнати із масиву кімнат. В залежності від розмірів згенерованої кімнати, та її пропорцій обирається кількість елементів освітлення та їх розташування. Алгоритм розтавляє ці елементи через рівні проміжки, та таким чином, щоб вони могли освітлювати всю площу кімнати. Код методу `AddLampToTheRoom` представлений нижче:

```
    void AddLampToTheRoom(int roomIndex, int lineCount, int
  rowCount)
        {
            Room room = dgCore.rooms[roomIndex];
```

```
            float length = room.verticalSize * oneStepSize;
            float widht = room.horizontalSize * oneStepSize;
            float deltaLength = length / (lineCount + 1);
            float deltaWidth = widht / (rowCount + 1);
            Vector3          point          =          new
  Vector3(dgCore.GetPoints(roomIndex)[0].x,              5f,
  dgCore.GetPoints(roomIndex)[0].z);
            for (int i = 0; i < lineCount; i++)
            {
                point.z += deltaLength;
                for (int y = 0; y < rowCount; y++)
                {
                    point.x += deltaWidth;
                    AddObjectToPoint(lamp, point);
                }
                point.x = dgCore.GetPoints(roomIndex)[0].x;
            }
        }
```

Алгоритм генерації освітлення в коридорах використовує метод `AddTorchToTheCoridor` для розміщення елементів освітлення вздовж стінок коридору. Елементи розташовані у шахмотному порядку, тобто перший елемент знаходиться на лівій стінці коридору, а другий елемент на правій і так далі, поки весь коридор не буде заповнений елементами освітлення. Відстані між кожним факелом залежать від розмірів самого коридору. Також алгоритмом передбаченно ситуацію, коли коридор занадто короткий і не потребує встановлення елементів освітлення в його межах.

Після того як було додано достатньо елементів освітлення на рівні, алгоритм переходить до заповнення кімнат елементами декору. Облаштування

кімнат має бути спеціально підготовленне завчасно гейм дизайнером, та готові елементи мають бути передані в алгоритм.

Алгоритм генераціх інтер'єру аналізує розміри кімнати, та її пропорції, відповідно до цього, обирає випадковий елемент з масиву елементів декору. Перевіряє чи площа цього об'єкту не перевищує двох третин площи кімнати, якщо перевірка пройшла, то встановлює єлемент інтер'єру на випадкову позицію в кімнаті, попередньо перевіривши чи він не виходить за рамки кімнати, та чи не перетинається з іншими об'єктами в кімнаті, якщо так, то вибирається інша випадкова позиція. Якщо обрана позиція відповідає всім вимогам, то елемент зберігається в масиві об'єктів які містить дана кімната. Якщо ж для обранного елементу ніяк не вдається встановити вірні координати, то цей елемент відкидується та з масиву обирається інший.

Після того, як були додані основні елементи інтер'єру, алгоритм переходить до останнього кроку, генерації не ігрових персонажів, для цього використовується метод `AddNPCToRoom`, який відповідно до розмірів кімнати, та вподобань гравця, від яких залежить кількість ворогів та їх рівень, обирає для кожного випадкове місцерозташування у межах кімнати. Алгоритм перевіряє чи обране місце знаходиться в кімнаті і чи його позиція не перетинається з іншими об'єктами в даній кімнаті, якщо так, то об'єкт створюється та додається в кімнату, якщо ж якась перевірка не пройшла, то обирається нове місцерозташування. Так повторюється поки алгоритм не знайде місце для потрібної кількості об'єктів, якщо ж місце в кімнаті закінчилось то алгоритм зупиняється та переходить до іншої кімнати. Після завершення роботи з однією кімнатою, алгоритм переходить до іншої.

На даному етапі робота алгоритму процедурної генерації рівня вважається повністю завершеною. Вихідним результатом є готовий згенерований рівень, відповідно до параметрів заданих підмодулем з елементами штучного інтелекту, або ж розробником. Рівень є завершеним, містить в собі визначену кількість кімнат, елементи освітлення, елементи інтер'єру, та об'єкти для взаємодої.

3.2 Реалізація підмодулю з елементами штучного інтелекту

Підмодуль штучного інтелекту відповідає за визначення вподобань гравця та підналаштуванню параметрів процедурної генерації рівня відповідно до стилю гри. В самому початку гри коефіцієнти налаштування параметрів мають стандартні значення. Під час проходження гравцем згенерованих рівнів коефіцієнти змінюються, а з ними змінюються і параметри процедурної генерації.

Чим довше грає гравець та чим більше разів він проходив згенеровані рівні, тим точніше підмодуль налаштовується на його вподобання. Перші проходження будуть сильно змінювати ваги коефіцієнтів, але зі збільшенням кількості ітерацій підлаштування зменшується відсоток змінення коефіцієнту. Таким чином алгоритм навчається в процесі роботи, як це роблять штучні нейронні сітки.

Параметри, які відстежує підмодуль з елементами штучного інтелекту:

1. Відсоток освоєння рівня
2. Час проходження битви
3. Кількість знайдених таємних кімнат

Відсоток освоєння рівня представляє собою різницю між загальною площею рівня та площею яку відкрив гравець по мірі свого проходження. Він визначається за допомогою матриці проходження. Під час переміщення по рівню в матрицю записується кожна точка яку відвідав гравець. Після проходження рівня матриця що була складена порівнюється з загальною матрицею рівня. Невідподність софрмованої матриці загальній і є ті невідкриті зони, в яких гравець не бував під час проходження. Визначається також середнє значення освоєння рівня, воно отримується у порівнянні всіх попередніх результатів проходжень. Також отримується коєфіцієнт останньої зміни параметру. Після отримання всіх значень алгоритм порівнює новий результат з середнім значенням і визначає коефіцієнт лінійності рівня. Якщо результати освоєння погіршились, тобто гравець був не сильно зацікавлений в проходженні рівня, то відповідно до останньої зміни приймається рішення

зменшити лінійність рівня, чи збільшити. Якщо ж результати освоєння покращились, тобто остання зміна коефіцієнту відповідала вподобанням гравця, то коефіцієнт змінюється відповідно до останньої зміни, але з меншим відсотком. Коефіцієнт лінійності впливає на параметри процедурної генерації, що в залежності від вподобань гравця будує більш лінійні рівні, або навпаки, рівні з великою кількістю кімнат, коридорів, та розгалуджень.

Час проходження битви це середній час, який знадобився гравцю для отримання перемоги над ворогом. Коли гравець проходить рівень, кожна його битва логується у відповідний файл, після проходження данні з файлу зчитуються і вираховується середній затрачений час на проходження битви у відповідному згенерованому рівні.

Тож, алгоритм вирахування коефіцієнту зміни складності:

1. Логування результатів битви під час проходження гравцем рівня;
2. Зчитування результатів логування та визначення середнього затраченого часу на проходження битви;
3. Отримання результатів попередніх проходжень інших рівнів;
4. Визначення відсотку зміни часу проходження (порівняння попередніх результатів з новим);
5. Налагодження коефіцієнту зміни складності (якщо результати значно погіршились коли коефіцієнт не був високим, то це означає, що гравець має труднощі з проходженням битв, отже рівень ворогів потрібно зменшити, навпаки якщо результати значно покращились, то це означає що битви занадто прості для даного гравця і потрібно підвищити складність).

Коефіцієнт зміни складності впливає на рівень ворогів, які розміщуються на рівні, та на їх кількість. Таким чином налаштовуються параметри які будуть оптимальні для проходження гравцем рівня, та які будуть поіністю відповідати його вподобанням та стилю гри.

Кількість знайдених таємних кімнат це параметр який відповідає кількості знайдених «секретів» під час проходження рівня. Таємні кімнати під час розробки розподіляються на три рівня складності. Спочатку для виявлення

вподобань гравця при генерації рівня додається стандартна кількість кімнат першого рівня складності. Під час проходження, кожний знайдений гравцем «секрет» записується і після проходження порівнюється кількість знайдених до загальної кількості. Якщо гравець знайшов всі кімнати які було сховано, то при наступному генеруванні рівня кількість схованих кімнат збільшується та їх складність випадковим чином збільшується. Так відбувається поки гравець продовжує знаходити всі таємні кімнати, та поки всі кімнати не будуть останнього, третього, рівня складності. Але якщо у гравця виникають складнощі з находженням «секретів», або вони йому просто не цікаві, кількість та рівень кімнат зменшується.

Таким чином, відстежуючи ці параметри під гри, підмодуль з елементами штучного інтелекту адаптується під конкретного гравця, його стиль гри та його вподобання. Враховуючи це, підмодуль змінює параметри процедурної генерації, намагаючись зробити так, щоб згенеровані рівні максимально підходили гравцю, що б викликало у нього ще більший інтерес до гри та до проходження процедурно згенерованих рівнів.

3.3 Модульне тестування

Модульні тести – це невеличкі програми, які дозволяють швидко протестувати окремі складові коду незалежно від інших частин застосування. Під час тестування вибираються частини коду, зазвичай це методи класів. Вони тестуються за допомогою відповідних засобів. Один з мінусів модульних тестів є те, що з часом їх стає надто багато. Тому процес проведення тестів автоматизують за допомогою відповідних фреймворків.

Виконаємо тестування для контролеру Controller в якому знаходяться методи методи `GetRoomToRoomClosestIndex` та `GetRandomHorizontal`, саме ці методи і будемо тестувати. Результат тестування зображено на рис. 3.5

Рисунок 3.5 – Результат виконання модульного тесту для контролеру `Controller`

Як ми можемо побачити на зображенні, тести виконалися успішно (відобразилися зелені позначки). У даному тесті виконувалася перевірка, чи метод `GetRoomToRoomClosestIndex` повертає коректне значення індексу найближчої кімнати (для тесту було створенно кімнати з строго заданими координатами). Також перевірявся метод `GetRandomHorizontal`, умовами проходження було те, що данний метод мав повернути корректне значення, яке б відповідало вимогам, а саме, значення повинно було лежати на проміжку горизонтальної межі завчасно згенерованої кімнати, та коридор з заданою шириною мав не виходити за межі кімнати.

Типи тестів представляються трьома етапами відповідно до номеру рядка: початкова ініціалізація контексту тестів, виконання дії яку треба протестувати та перевірка, чи була виконана відповідна умова.

Додамо подібні тести для деяких методів класу `Room`, результати тестування можна побачити на рис. 3.6

Рисунок 3.6 – Результат виконання модульного тесту для сторінки для класу `Room`

Як ми можемо пересвідчитись, модульне тестування є дуже корисним методом для перевірки реакції системи на різні вхідних даних, що дозволяє зекономити час розробника та провести якісне тестування.

Розглянемо аглоритм процедурної генерації рівня детальніше:

1.Генерування схематичного відображення кімнат та коридорів. Алгоритм генерації створює відповідно до заданих параметрів конкретну

кількість кімнат та проводить між ними коридори, поки це створює лише на схематичному рівні, генерується лише каркас кімнати яка буде збудована(рис. 3.7).

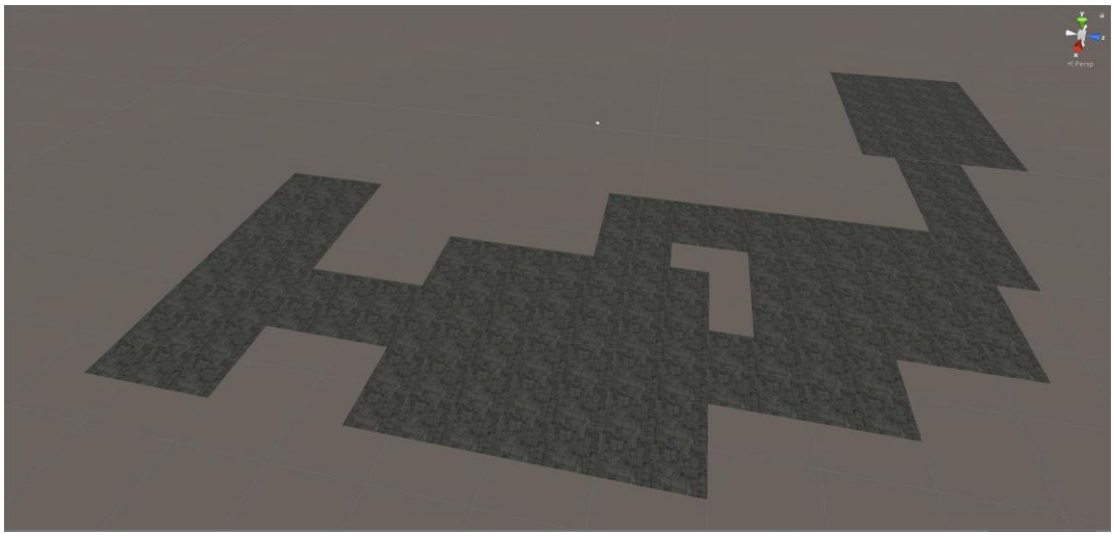


Рисунок 3.7 – Схематичне зображення згенованого рівня

2. Створення стін

Алгоритм генерації проходить по матриці проходимості, та відповідно до значень генерує підлогу, стелю та стіни рівня, елементи для генерації мають бути задані передчасно гейм дизайнером, та мають відповідати тематиці та антуражу рівня. Для кращого відображення роботи алгоритму було вирішено зробити стелю прозорою. Згенерований рівень із стінами можна побачити на рис. 3.8.

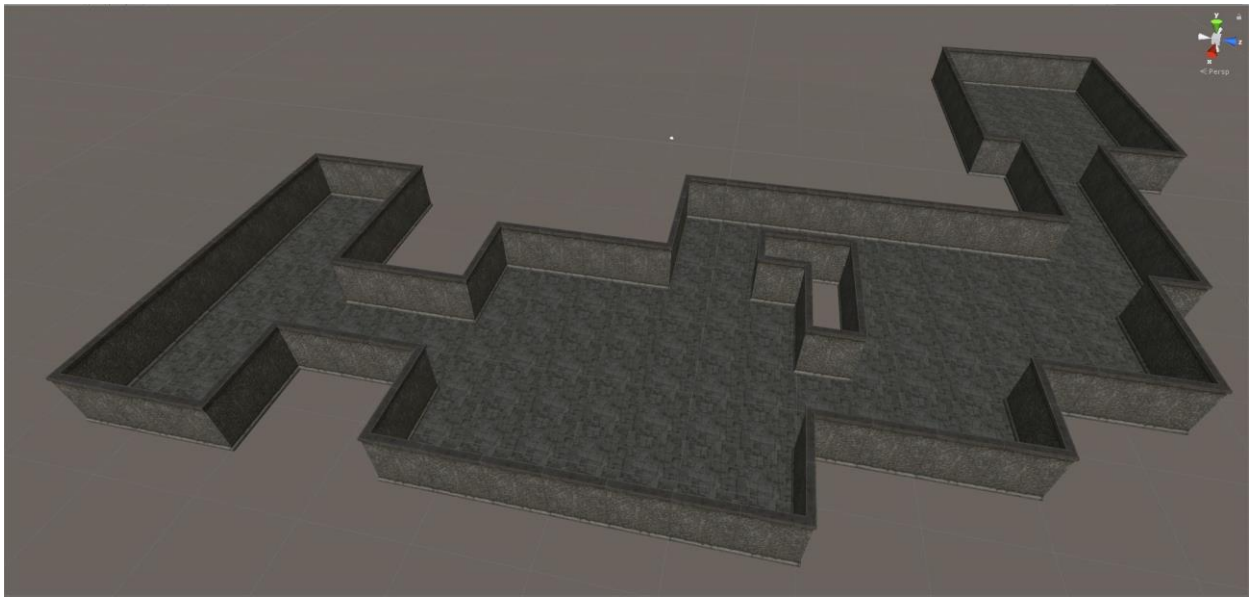


Рисунок 3.8 – Згенерований рівень з створеними стінами

3. Розміщення елементів освітлення в кімнатах

Після створення основи рівня, алгоритм переходить до розміщення елементів освітлення в кожній кімнаті(рис. 3.9). Вираховуються рівні проміжки, через які потрібно встановити елементи освітлення для кожної конкретної кімнати. Після цього генерується певна кількість світильників для кожної кімнати, в залежності від її розмірів. Елементи освітлення що будуть розміщатись мають бути задані завчасно гейм дизайнером.

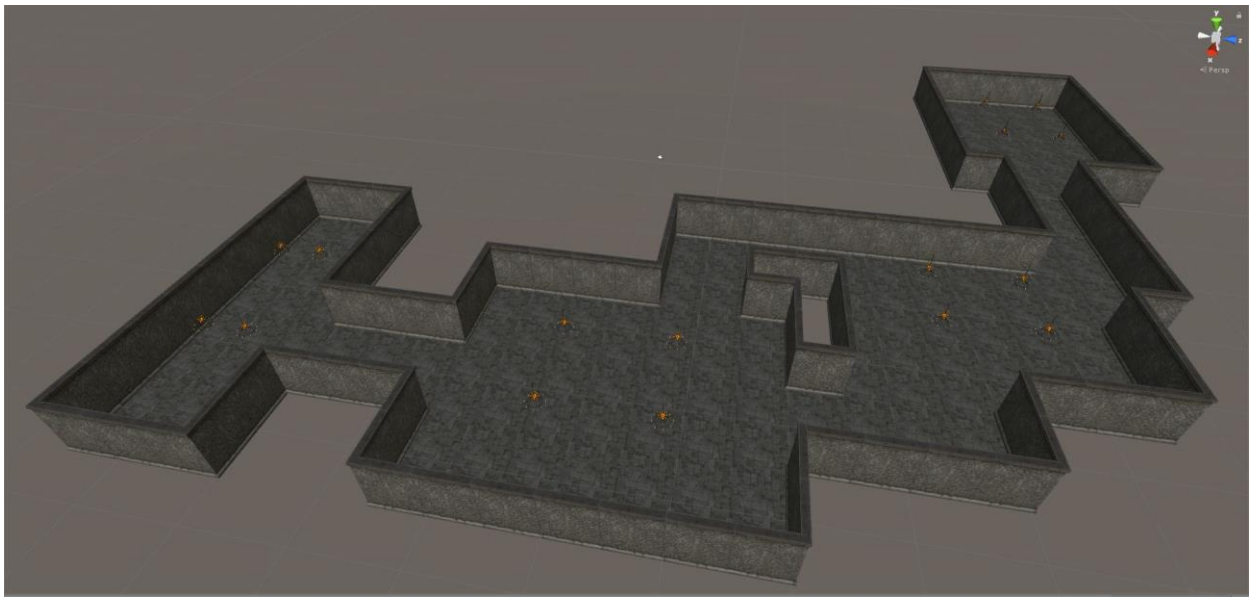


Рисунок 3.9 – Згенерований рівень з елементами освітлення в кімнатах

4. Розміщення елементів освітлення в коридорах

Аналогічно як і для кімнат, створюється освітлення і для коридорів, що з’єднують ці кімнати. Елементи освітлення генеруються через рівні проміжки, що визначаються в залежності від довжини коридору, та розміщуються у «шахматному» порядку.

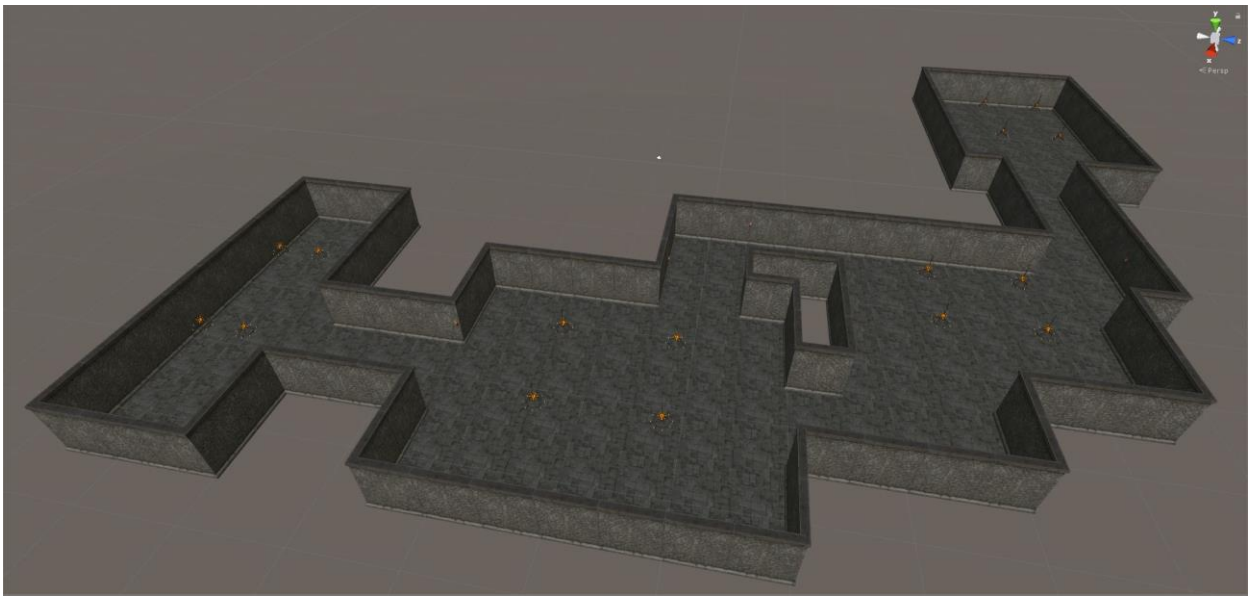


Рисунок 3.10 – Згенерований рівень з елементами освітлення в кімнатах та коридорах

5. Розміщення елементів інтер’єру в кімнатах

Після завершення генерації кімнат та елементів освітлення в них, алгоритм переходить до генерації інтер’єру в створених кімнатах. Базові шаблони інтер’єру кімнат мають бути зроблені гейм дизайнером та передані алгоритму завчасно. Алгоритм в залежності від розмірів кімнати, її пропорції підбирає випадковий відповідний шаблон розміщення з переданих. Результати цього процесу для однієї кімнати можна побачити на рис. 3.11.

Рисунок 3.11 – Згенерована кімната з елементами інтер’єру

6. Розміщення NPC в кімнатах

Останнім етапом процедурної генерації є розміщення NPC в кімнатах рівня. В залежності від розмірів кімнати та шаблону інтер’єру що був використанний для неї, встановлюється максимальна кількість ворогів. NPC розміщуються завдяки випадковому вибору точки в кімнаті, далі алгоритм перевіряє чи дана точка не є зайнята, та чи об’єкт розміщенний в цій точці не бути перетинатись з іншими об’єктами в кімнаті. Коли всі перевіркуи було пройдено NPC розміщається у відповідну точку. Результат роботи алгоритму можна побачити на рис. 3.12.

Рисунок 3.12 – Повністю згенерована кімната

Висновки до розділу

В даному розділі:

1) Розроблені допоміжні класи, розглянто їх структуру та основні методи.

2) Розроблено підсистему процедурної генерації.

3) Розроблено підсистему розміщення об’єктів, генерації освітлення та інтер’єру.

4) Детально розглянуто структуру та алгоритми розроблених підсистем.

5) Ророблено та проаналізовано підмодуль з елементами штучного інтелекту.

6) Проведено тестування системи. Для тестування використовувалось ручне та модульне тестування.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

Повністю безпечних і нешкідливих виробництв не існує. Завдання охорони праці — звести до мінімуму ймовірність нещасного випадку або захворювання працюючого з одночасним забезпеченням комфортних умов при максимальній продуктивності праці.

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці.

Нерозривно з охороною праці пов'язані і питання пожежної безпеки. Для охорони праці важливе значення мають стан повітряного середовища виробничих приміщень, їх освітлення, вентиляція, електромагнітні і радіоактивні випромінювання, вібрація і шум.

4.1. Характеристика робочого приміщення

Таблиця 4.1 — Параметри робочого приміщення

№	**Назва**	**Характеристика**
1	Ширина	7 м
2	Довжина	8 м
3	Висота	2.8 м
4	Підлога	Ламінат
5	Стіни	Цегла
6	Покриття стін	Обої світлих тонів
7	Природнє освітлення	Два вікна на північ
8	Штучне освітлення	12 світильників (люстра з плафоном), що мають 3 галогенні лампи по 100 Вт
9	Вологість	сухо
10	Опалення	водяне
11	Електропроводка	прихована, трьохпровідна
12	Вентиляція	загально-обмінна вентиляція
13	Вологість	Не більше 75%

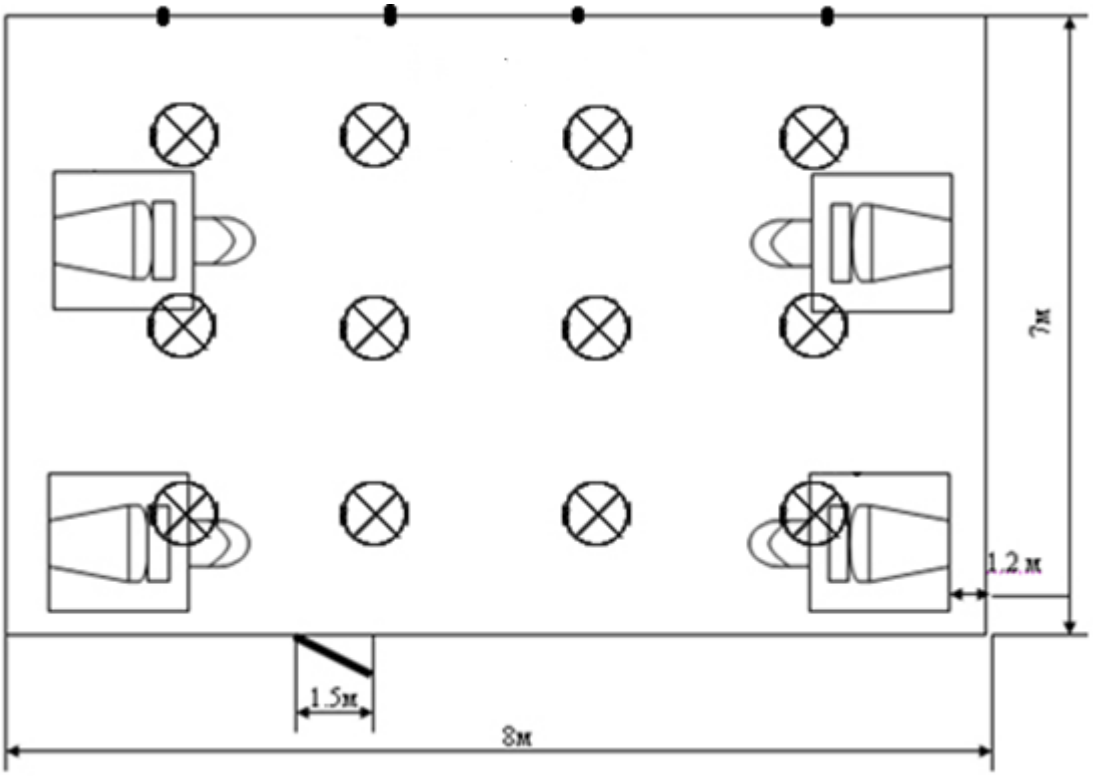
Площа приміщення:

$$S = A*B = 56 \text{ (м}^2\text{)}.$$

Обсяг приміщення:

$$V = S \square h = 56*2,8 = 156,8 \text{ (м}^3\text{)}.$$

Кількість постійних працівників у приміщенні - 4 чоловік.



Таблиця 4.2 — Параметри робочого місця

№	Просторові параметри L	мм
1	Висота сидіння	400-500
2	Висота клавіатури від підлоги	600-750
3	Кут нахилу клавіатура	7-15 °
4	Ширина основної клавіатура залишиться	> 400
5	Глибина основний клавіатура залишиться	> 200
6	Видалення клавіатури від краю столу	80-100
7	Висота екрану від рівня підлоги	950-1000
8	Кут нахилу екрану і нормалі	0-30 °
9	Відстань екрану від краю столу	500-700
10	Висота поверхонь для записів	670-850

Закінчення таблиці 4.2

11	Площа поверхні для записів	600х400
12	Кут нахилу поверхні для записів	0-100
13	Глибина простору для ніг у колінах	<400
14	Глибина простору на рівні ступень	<600
15	Висота простору для ніг у колінах	<600
16	Висота простору на рівні ступень	<100
17	Ширина простору для ніг на рівні	<500
18	Висота підставки для ніг	50-130
19	Кут підставки для ніг	0-25
20	Ширина підставки для ніг	300
21	Глибина підставки для ніг	400

4.2 Аналіз шкідливих небезпечних виробничих факторів

4.2.1 Мікроклімат робочого приміщення

Параметри мікроклімату можуть мінятися в широких межах, у той час як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату — створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

В даному приміщенні встановлена категорія роботи легка 1а. До неї відносяться роботи, вироблені сидячи, з незначними фізичними вправами.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих приміщень встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються

залежно від пори року, характеру трудового процесу і характеру виробничого приміщення.

Таблиця 4.3 — Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні	22 ... 24 ° С
	Відносна вологість	40 ... 60%
	Швидкість руху повітря	до 0,1 м / с
Теплий	Температура повітря в приміщенні	23 ... 25 ° С
	Відносна вологість	40 ... 60%
	Швидкість руху повітря	0,1 ... 0,2 м / с

Для регулювання температури в різні пори року, в приміщенні повинні бути встановлені кондиціонер і опалення.

4.2.2 Шум і вібрація в робочих приміщеннях

Шум погіршує умови праці здійснюючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах і т. Д. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму [вище 80 дБ (А)] на слух людини приводить до його часткової або повної втрати.

Рівень шуму в залах обробки інформації на обчислювальних машинах - 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані звукопоглинальними матеріалами.

Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

Таблиця 4.4 — Джерела шуму

Джерело шуму	**Рівень шуму, дБА**
Жорсткий диск	40
Вентилятор	45
Монітор	17
Клавіатура	10
Принтер	45
Сканер	42

Максимальний час робота принтера за один день — 2,5 години.

Робочий день Т = 8 годин.

Час роботи приводів - півгодини.

$$L_{\Sigma}=10\cdot\lg(10^{4}+10^{4,5}+10^{1,7}+10^{1}+10^{4,5}+10^{4,2})=49,5 \text{ дБА}$$

Гранично допустимі норми встановлені в ДНС 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку.

Що не перевищує норму в 50 дБА. Таким чином, акустичне середовище приміщень знаходиться в межах оптимальних величин.

4.2.3 Аналіз освітлення

Природне і штучне освітлення нормується в ДБН В.2.5-28-2006 Природне і штучне освітлення залежно від характеристики зорової роботи, найменшого розміру об'єкта розрізнення, фону контрасту об'єкта з фоном. Природне освітлення в аналізованому приміщенні реалізується за допомогою системи бічного освітлення, що складається з двох вікон (2х 2м х 1,5м). Перед вікнами аналізованого приміщення немає протиборчих будинків, тому сонячний світло не затінюється і повністю потрапляє в дане.

Нормоване значення КПО. Таким чином, розрахункове значення КПО длжно равнятся або бути більше нормованого. Тоді зорові умови праці в

досліджуваному приміщенні при природному освітленні можна вважати задовільними.

У розглянутому приміщенні використовується система загального штучного освітлення. Є 12 галогенних ламп розжарювання фірми Phlips Reflector lamp (1x13677-38-35W). Враховуючи характер системи освітлення для розрахунку використовуємо метод коефіцієнта використання світлового потоку .Висота підвісу над робочою поверхнею складає 2 м. Видалення від точки підвісу до робочих місць становить 1,8м.

Відповідає нормі Ен = 300 для роботи з монітором, і для роботи з документацією Ен = 400лк.

4.3 Електробезпека

Споживачі електроенергії в приміщенні - освітлювальні прилади та обчислювальна техніка. У розглянутій лабораторії використовується однофазна мережа електропостачання з напругою 220 В і частотою 50 Гц. Проводка проведена прихованим способом, корпуси системних блоків комп'ютерів підключені до системи занулення (через трьохполюсний роз'єм євровилку), світильники розташовані на висоті 2,6. м, що перевищує висоту 2.5 м, встановлену, тому виключена можливість дотику персоналу до оголених проводів.

Приміщення являє собою сухе, безпилове приміщення з нормальною температурою повітря і ізолюючим підлогою. Кількість заземлених предметів невелика (дві батареї водяного опалення). Відсутня можливість одночасного контакту корпусу приладу (системний блок ПК) і заземленою конструкції. Приміщення за ступенем небезпеки ураження електричним струмом відноситься до приміщень без підвищеної небезпеки. Все обладнання відповідає міжнародному електротехнічного стандарту РЄ і допускається до застосування в побутових умовах. Усі заходи щодо забезпечення електробезпеки в приміщенні радять вимогам ДНАОП 0.00-1-31-99 Правила охорони праці під час експлуатації електронно-обчислювальних машин. Всі

працівники приміщення ознайомлені з правилами техніки безпеки при роботі з електрообладнанням

4.4 Пожежна безпека

У розглянутому приміщенні знаходяться ЕОМ, в яких дуже висока щільність розміщення елементів електронних схем. Сама ЕОМ являє собою пожежну небезпеку, так як при підвищенні температури окремих вузлів можливо оплавлення ізоляції сполучних проводів, яке веде до короткого замикання, що супроводжується, в свою чергу, іскрінням.

Категорія пожежної безпеки даного приміщення - В, помірна пожежонебезпека.

Можливі причини пожежі:

а) перевантаження в електромережі;

б) коротке замикання;

в) руйнування ізоляції провідників.

Система протипожежного захисту:

а)встановлено автоматична пожежна сигналізація на димових сповіщувачах ДІП-1, з розрахунку 2 шт. на кожні 20 м2 площі приміщення, враховуючи високу вартість обладнання, наявність прихованих комунікацій і специфіку загоряння ЕОМ. Тобто на площу 56 м2 необхідно 6 димових сповіщувача.

б) розміщені 6 вуглекислотних вогнегасники ОУ-5 (ручні) з розрахунку 1 вогнегасник на 10 м2.

Організаційні заходи:

а)проводиться інструктаж персоналу по ТБ;

б)розроблені заходи щодо дій адміністрації на випадок виникнення пожежі;

в)схема евакуації при пожежі поміщена на видному місці;

г)ширина дверного отвору на випадок евакуації 1,5 м., висота 2 м.

6.5 Інструкція з техніки безпеки

До роботи на персональній електронно-обчислювальній машині допускаються після вивчення даної інструкції особи, які пройшли попередній медичний огляд, встановлений курс навчання за даною професією, пройшли вступний та первинний інструктажі з питань охорони праці, пожежної безпеки, інструктаж і перевірку знань з електробезпеки і отримали II кваліфікаційну групу.

Перед початком роботи на ПК користувач повинен:

а)пересвідчитися у цілості корпусів і блоків (обладнання) ПК;

б)перевірити наявність заземлення, справність і цілість кабелів живлення, місця їх підключення.

Не рекомендується вмикати ПК та починати роботу при виявлених несправностях. Під час роботи пересвідчившись у справності обладнання, не допускати у зону сторонніх осіб, увімкнути електроживлення ПК, розпочати роботу, дотримуючись умов інструкції з її експлуатації.

Не рекомендується:

а)замінювати змінні елементи або вузли та проводити перемонтаж при ввімкненому ПК;

б)з'єднувати і роз'єднувати вилки та розетки первинних мереж електроживлення, які знаходяться під напругою;

в)знімати кришки, які закривають доступ до струмопровідних частин мережі первинного електроживлення при ввімкненому обладнанні;

г)користуватися паяльником з незаземленим корпусом;

д)замінювати запобіжники під напругою;

е)залишати ПК у ввімкненому стані без нагляду.

По закінченні робочого дня:

а)кнопкою "ВИМК" відключити електроживлення ПК згідно з інструкцією експлуатації, вийнявши вилку кабелю живлення з розетки;

б) впорядкувати робоче місце користувача ПК, прибравши використане обладнання та матеріали у відведені місця;

в) про виявлені недоліки у роботі ПК протягом робочого часу необхідно повідомити відповідним посадовим особам та спеціалістам.

Залишаючи приміщення після закінчення робочого дня, дотримуючись встановленого режиму огляду приміщення, необхідно:

а) зачинити вікна, кватирки;

б) перевірити приміщення на відсутність тліючих предметів;

в)відключити від електромережі всі електроприлади, електрообладнання та вимкнути освітлення;

г) зачинити вхідні двері приміщення на замок і ключ здати черговому охорони.

Висновок до розділу

У ході розробки розділу вивчено приміщення, в якому розроблювалося програмне забезпечення. Описано та проаналізовано небезпечні й шкідливі виробничі фактори у виробничому приміщенні з радіоелектронною апаратурою.. У результаті проведення робіт встановлено, що об’єм і площа приміщення, які відводяться на одного працівника, відповідають нормативним значенням. Для цих факторів визначено нормативні показники у відповідності з діючою нормативно- технічною документацією та проведено порівняння з фактичними їх значеннями. З виявлених небезпечних та шкідливих факторів виділено найбільш несприятливі для яких було розроблено детальні заходи з охорони праці. Аналіз усіх розрахованих у даному розділі факторів показав результати, які дають всі підстави вважати, що розглянуте виробниче приміщення повністю відповідає всім нормативним документам і вимогам.

ВИСНОВКИ

Дипломний проект присвячена розробці модулю процедурної генерації рівня RPG гри з використанням штучного інтелекту. Існуючі алгоритми процедурної генерації розробляються виключно під конкретну гру, зазвичай їх не можливо застосувати до інших ігор, а модуль штучного інтелекту для визначення вподобання гравця робить алгоритм менш випадковим та більш налаштованим на конкретного гравця. Враховуючи вищезазначену ситуацію створення модулю процедурної генерації з елементами штучного інтелекту є актуальним напрямком розробки.

Програмна частина системи була реалізована з застосуванням сучасних рішень та технологій розробки, а правильність роботи була перевірена в ході тестування програмного продукту.

Був розроблений незалежний модуль, який може бути використаний як для автоматичної генерації рівнів під час гри, так і як інструмент генерації базової структури рівня при роботі гейм дизайнеру. Підмодуль з використанням штучного інтелекту може під час гри виявляти вподобання гравця щодо згенерованого рівня та підлаштовувати роботу алгоритму процедурної генерації під вимоги конкретного гравця. Система працює корректно та може використовуватись як модуль для різніх ігрів та жанрів завдяки своїй гнучкості та маштабуємості.

ПЕРЕЛІК ПОСИЛАНЬ

1. Вікіпедія. Процедурна генерація [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Процедурная_генерация
2. Вікіпедія. .NET [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/.NET
3. Вікіпедія. C Sharp [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp
4. Вікіпедія. ASP.NET [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/ASP.NET
5. Вікіпедія. ASP.NET MVC Framework [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/ASP.NET_MVC_Framework
6. Unity. [Електронний ресурс] - https://ru.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA)
7. Unreal Engine. [Електронний ресурс] - https://ru.wikipedia.org/wiki/Unreal_Engine
8. C#. [Електронний ресурс] - https://ru.wikipedia.org/wiki/C_Sharp
9. JavaScript. [Електронний ресурс] - https://ru.wikipedia.org/wiki/JavaScript
10. Microsoft Visual Studio. [Електронний ресурс] - https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio
11. SharpDevelop. [Електронний ресурс] - https://ru.wikipedia.org/wiki/SharpDevelop
12. Штучний інтелект бойової системи гри. [Електронний ресурс] - http://vco.su/gadzhety/iskusstvennyy-intellekt-v-kompyuternykh-igrakh
13. Модульне тестування. [Електронний ресурс] - https://habr.com/post/169381/

ДОДАТОК А

Вихідні коди класа контролера «CreatorController»

```
public void Init(int _dSize, int _roomSize, int _roomSizeDelta, int _roomsCount, bool
_isIntersections, int _coridorThickness, float _stepSize, float _whProp, int
_coridorsCount)
    {
        dSize = _dSize;
        roomSize = _roomSize;
        roomSizeDelta = _roomSizeDelta;
        roomsCount = _roomsCount;
        coridorThicknes = _coridorThickness;
        stepSize = _stepSize;
        whProp = _whProp;
        coridorsCount = _coridorsCount;

        ClearGeneratedWalls();
    }

    public bool isCorrect()
    {
        if (rooms != null && rooms.Count > 1)
        {
            return true;
        }
        return false;
    }

    public void Generate()
    {
        int generatedRooms = 0;
        int genStep = 0;
        rooms = new List<Room>();
        coridors = new List<Room>();
        while (generatedRooms < roomsCount && genStep < generationCycleLimits)
        {
            Room newRoom = GenerateRandomRoom();
            if (!newRoom.IsTooSmall(coridorThicknes) && !newRoom.IsIntersect(rooms))
            {
                rooms.Add(newRoom);
                generatedRooms++;
            }
            else
            {
                genStep++;
            }
        }

        if (rooms.Count > 1)
        {//есть хотя бы две комнаты, строим коридоры
            //для каждой ищем ближайшую
            for (int i = 0; i < rooms.Count; i++)
            {
                //int j = GetRoomToRoomClosestIndex(i);
                int[] toRoomClostArray = GetRoomToRoomClosestIndexesArray(i);
                for (int j = 0; j < coridorsCount; j++)
                {
                    if (j < toRoomClostArray.Length)
                    {
                        int toRoomId = toRoomClostArray[j];
                        if (toRoomId != i && toRoomId > -1)
                        {
```

```
                            BuildCoridor(i, toRoomId);
                        }
                    }
                }
            }

            //дальше формируем матрицу с проходимостью
            PointPair minMax = GetMinMax();
            dgMap = new Map(minMax.secondPoint.GetY() - minMax.firstPoint.GetY(),
minMax.secondPoint.GetX() - minMax.firstPoint.GetX(), minMax.firstPoint,
minMax.secondPoint);
            foreach (Room room in rooms)
            {
                dgMap.FillByRoom(room, false);
            }
            foreach (Room coridor in coridors)
            {
                dgMap.FillByRoom(coridor, true);
            }

        }


    }

    PointPair GetMinMax()
    {
        PointPair toReturn = new PointPair();
        Point rMin = rooms[0].GetCorner(0);
        Point rMax = rooms[0].GetCorner(2);
        toReturn.firstPoint = new Point(rMin);
        toReturn.secondPoint = new Point(rMax);
        foreach (Room room in rooms)
        {
            rMin = room.GetCorner(0);
            rMax = room.GetCorner(2);
            if (rMin.GetX() < toReturn.firstPoint.GetX())
            {
                toReturn.firstPoint.SetX(rMin.GetX());
            }
            if (rMin.GetY() < toReturn.firstPoint.GetY())
            {
                toReturn.firstPoint.SetY(rMin.GetY());
            }

            if (rMax.GetX() > toReturn.secondPoint.GetX())
            {
                toReturn.secondPoint.SetX(rMax.GetX());
            }
            if (rMax.GetY() > toReturn.secondPoint.GetY())
            {
                toReturn.secondPoint.SetY(rMax.GetY());
            }
        }

        foreach (Room coridor in coridors)
        {
            rMin = coridor.GetCorner(0);
            rMax = coridor.GetCorner(2);
            if (rMin.GetX() < toReturn.firstPoint.GetX())
            {
                toReturn.firstPoint.SetX(rMin.GetX());
            }
            if (rMin.GetY() < toReturn.firstPoint.GetY())
            {
```

```
                toReturn.firstPoint.SetY(rMin.GetY());
            }

            if (rMax.GetX() > toReturn.secondPoint.GetX())
            {
                toReturn.secondPoint.SetX(rMax.GetX());
            }
            if (rMax.GetY() > toReturn.secondPoint.GetY())
            {
                toReturn.secondPoint.SetY(rMax.GetY());
            }
        }

        //Debug.Log("Minimum: " + toReturn.point01.ToString() + " Maximum: " +
toReturn.point02.ToString());

        return toReturn;
    }

    void BuildCoridor(int startIndex, int endIndex)
    {
        Room startRoom = rooms[startIndex];
        Room endRoom = rooms[endIndex];
        int sPos = startRoom.GetRandomVertical(coridorThicknes);
        int ePos = endRoom.GetRandomHorizontal(coridorThicknes);
        int clostV = startRoom.GetClosestVerticalCoordinate(ePos);
        int clostH = endRoom.GetClosestHorizontalCoordinate(sPos);

        Room newCoridor01 = new Room(new Point(clostV, sPos - coridorThicknes / 2), new
Point(ePos + (Sign(ePos - clostV) > 0 ? coridorThicknes - coridorThicknes / 2 : -1 *
coridorThicknes / 2), sPos - coridorThicknes / 2 + coridorThicknes));
        Room newCoridor02 = new Room(new Point(ePos - coridorThicknes / 2, clostH), new
Point(ePos - coridorThicknes / 2 + coridorThicknes, sPos + (Sign(sPos - clostH) > 0 ?
coridorThicknes - coridorThicknes / 2 : -1 * coridorThicknes / 2)));

        coridors.Add(newCoridor01);
        coridors.Add(newCoridor02);

    }

    int Sign(int i)
    {
        if (i >= 0)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }

    float Sign(float i)
    {
        if (i >= 0)
        {
            return 1f;
        }
        else
        {
            return -1f;
        }
    }

    int GetRoomToRoomClosestIndex(int i)
```

```
    {
        float closestDistance = 2 * dSize;
        Room room = rooms[i];
        int closestIndex = -1;
        for (int j = 0; j < rooms.Count; j++)
        {
            if (i != j)
            {
                float d = room.GetCornerDistance(rooms[j]);
                if (d < closestDistance)
                {
                    closestDistance = d;
                    closestIndex = j;
                }
            }
        }

        return closestIndex;
    }
    int[] GetRoomToRoomClosestIndexesArray(int id)
    {
        Dictionary<int, float> roomDistanceMap = new Dictionary<int, float>();
        for (int i = 0; i < rooms.Count; i++)
        {
            if (i != id)
            {
                float d = rooms[id].GetCornerDistance(rooms[i]);//расстояние от текущей
комнаты до i-ой
                roomDistanceMap.Add(i, d);
            }
        }
        //Debug.Log(id.ToString() + " room: " + dictToString(roomDistanceMap));
        int[] toReturn = new int[roomDistanceMap.Count];
        float recordMin = -1f;
        int recordMinKey = -1;
        for (int i = 0; i < toReturn.Length; i++)
        {
            float min = 0f;
            int minKey = -1;
            foreach (int k in roomDistanceMap.Keys)
            {
                if (minKey == -1)
                {
                    if (roomDistanceMap[k] >= recordMin && k != recordMinKey)
                    {
                        min = roomDistanceMap[k];
                        minKey = k;
                    }
                }
                else
                {
                    if (roomDistanceMap[k] > recordMin && roomDistanceMap[k] < min)
                    {
                        min = roomDistanceMap[k];
                        minKey = k;
                    }
                }
            }
            toReturn[i] = minKey;
            recordMin = min;
            recordMinKey = minKey;
        }
        //Debug.Log(id.ToString() + " room: " + arrayToString(toReturn));

        return toReturn;
```

```
    }

    string arrayToString(int[] array)
    {
        string str = "";
        for (int i = 0; i < array.Length; i++)
        {
            str += array[i] + " ";
        }

        return str;
    }

    string dictToString(Dictionary<int, float> dict)
    {
        string str = "";
        foreach (int k in dict.Keys)
        {
            str += k.ToString() + ": " + dict[k].ToString() + "; ";
        }
        return str;
    }

    float FilterRandom(bool isWidth, float value)
    {
        if (isWidth)
        {
            return value * Mathf.Pow(propCoefficient, whProp);
        }
        else
        {
            return value * Mathf.Pow(propCoefficient, -1 * whProp);
        }
    }

    Room GenerateRandomRoom()
    {
        float xRandom = Random.Range(-1f, 1f);
        float yRandom = Random.Range(-1f, 1f);

        int cx = (int)(Sign(xRandom) * dSize * FilterRandom(true, Mathf.Abs(xRandom)) /
(2 * Mathf.Pow(propCoefficient, Mathf.Abs(whProp))));
        int cy = (int)(Sign(yRandom) * dSize * FilterRandom(false, Mathf.Abs(yRandom)) /
(2 * Mathf.Pow(propCoefficient, Mathf.Abs(whProp))));

        int w = Random.Range(roomSize - roomSizeDelta, roomSize + roomSizeDelta);
        int h = Random.Range(roomSize - roomSizeDelta, roomSize + roomSizeDelta);

        return new Room(cx, cy, w, h);
    }

    public Vector3[] GetPoints(int roomIndex)
    {
        Vector3[] a = rooms[roomIndex].GetPointsArray();
        Vector3[] toReturn = new Vector3[a.Length];
        for (int i = 0; i < a.Length; i++)
        {
            toReturn[i] = a[i] * stepSize;
        }
        return toReturn;
    }

    public Vector3[] GetCoridorPoints(int coridorIndex)
    {
        Vector3[] a = coridors[coridorIndex].GetPointsArray();
```

```
        Vector3[] toReturn = new Vector3[a.Length];
        for (int i = 0; i < a.Length; i++)
        {
            toReturn[i] = a[i] * stepSize;
        }
        return toReturn;
    }

    public int GetCoridorsCount()
    {
        return coridors.Count;
    }

    public int GetRoomsCount()
    {
        return rooms.Count;
    }

    void ClearGeneratedWalls()
    {
        if (wallModels.Count > 0)
        {
            for (int i = wallModels.Count - 1; i > -1; i--)
            {
                if (wallModels[i] != null)
                {
                    DestroyImmediate(wallModels[i]);
                }
            }
        }
        if (rootObject != null)
        {
            DestroyImmediate(rootObject);
        }
        if (floorRoot != null)
        {
            DestroyImmediate(floorRoot);
        }
        if (wallsRoot != null)
        {
            DestroyImmediate(wallsRoot);
        }
    }

    void EmitElement(GameObject element, Vector3 position, bool isFloor, int id, bool
isColorize)
    {
        if (element != null)
        {
            GameObject newInst = (GameObject)PrefabUtility.InstantiatePrefab(element);
            newInst.transform.position = position;
            newInst.GetComponent<Renderer>().material = new
Material(newInst.GetComponent<Renderer>().sharedMaterial);
            DRId drIdComponent = newInst.GetComponent<DRId>();
            if (drIdComponent != null)
            {
                drIdComponent.id = id;
            }
            if (isColorize)
            {
                SetColor(id, newInst);
            }
            wallModels.Add(newInst);
            if (!isFloor)
            {
```

```
                newInst.transform.SetParent(wallsRoot.transform);
            }
            else
            {
                newInst.transform.SetParent(floorRoot.transform);
            }

        }
    }

    void SetColor(int id, GameObject go)
    {
        if (id <= -1)
        {
            go.GetComponent<Renderer>().sharedMaterial.color = new Color(227f / 255f,
227f / 255f, 227f / 255f);
        }
        else
        {
           // go.GetComponent<Renderer>().sharedMaterial.color =
DRCustomEditor.HSVToRGB(DRCustomEditor.GetSin(id, 0f), 0.5f, 0.8f);
        }
    }

    int SetId(int id, bool isSet)
    {
        if (isSet)
        {
            return id;
        }
        else
        {
            return -1;
        }
    }

    public void EmitGeometry(GameObject lineLGO, GameObject lineRGO, GameObject lineTGO,
GameObject lineBGO, GameObject ICornerTLGO, GameObject ICornerTRGO, GameObject
ICornerBLGO,
        GameObject ICornerBRGO, GameObject OCornerTLGO, GameObject OCornerTRGO,
GameObject OCornerBLGO, GameObject OCornerBRGO, GameObject floorPlate, float stepSize,
bool isSetId)
    {
        ClearGeneratedWalls();
        wallModels = new List<GameObject>();
        rootObject = new GameObject();
        rootObject.name = rootName;
        floorRoot = new GameObject();
        floorRoot.transform.SetParent(rootObject.transform);
        floorRoot.name = floorRootName;
        wallsRoot = new GameObject();
        wallsRoot.transform.SetParent(rootObject.transform);
        wallsRoot.name = wallsRootName;

        if (dgMap != null)
        {
            int u = dgMap.GetRowMax();
            int v = dgMap.GetColumMax();
            for (int i = 0; i < u + 1; i++)
            {
                for (int j = 0; j < v + 1; j++)
                {
                    bool center = dgMap.GetCell(i, j);
                    bool left = dgMap.GetCell(i, j - 1);
                    bool top = dgMap.GetCell(i - 1, j);
```

```
                    bool leftTop = dgMap.GetCell(i - 1, j - 1);
                    Vector3 coordinate = stepSize * dgMap.GetPosition(i, j);
                    if (center)
                    {//открытая клетка
                        EmitElement(floorPlate, stepSize * dgMap.GetCenterPosition(i, j),
true, SetId(0, isSetId), isSetId);
                        EmitElement(floorPlate, new Vector3(coordinate.x+3, 6.05f,
coordinate.z-3), true, SetId(0, isSetId), isSetId);
                        if (left && leftTop && top)
                        {

                        }
                        else if (left && leftTop && !top)
                        {
                            coordinate.x = coordinate.x + 5;
                            EmitElement(OCornerBLGO, coordinate, false, SetId(5,
isSetId), isSetId);
                        }
                        else if (left && !leftTop && top)
                        {
                            coordinate.z = coordinate.z + 5;
                            EmitElement(OCornerBRGO, coordinate, false, SetId(8,
isSetId), isSetId);
                        }
                        else if (left && !leftTop && !top)
                        {
                            EmitElement(lineTGO, stepSize * dgMap.GetPosition(i, j),
false, SetId(10, isSetId), isSetId);
                        }
                        else if (!left && leftTop && top)
                        {
                            coordinate.x = coordinate.x - 5;
                            EmitElement(OCornerTRGO, coordinate, false, SetId(7,
isSetId), isSetId);
                        }
                        else if (!left && leftTop && !top)
                        {
                            coordinate.z = coordinate.z - 5;
                            //спорный случай
                            EmitElement(ICornerTLGO, coordinate, false, SetId(2,
isSetId), isSetId);
                            coordinate.z = coordinate.z + 10;
                            EmitElement(ICornerBRGO, coordinate, false, SetId(4,
isSetId), isSetId);
                        }
                        else if (!left && !leftTop && top)
                        {
                            EmitElement(lineLGO, stepSize * dgMap.GetPosition(i, j),
false, SetId(9, isSetId), isSetId);
                        }
                        else if (!left && !leftTop && !top)
                        {
                            coordinate.z = coordinate.z - 5;
                            EmitElement(ICornerTLGO, coordinate, false, SetId(2,
isSetId), isSetId);
                        }
                    }
                    else
                    {
                        if (left && leftTop && top)
                        {
                            coordinate.z = coordinate.z - 5;
                            EmitElement(OCornerTLGO, coordinate, false, SetId(6,
isSetId), isSetId);
                        }
```

```
                        else if (left && leftTop && !top)
                        {
                            EmitElement(lineRGO, stepSize * dgMap.GetPosition(i, j),
    false, SetId(11, isSetId), isSetId);
                        }
                        else if (left && !leftTop && top)
                        {
                            coordinate.z = coordinate.z + 5;
                            //спорный случай
                            EmitElement(OCornerBRGO, coordinate, false, SetId(8,
    isSetId), isSetId);
                            coordinate.z = coordinate.z - 10;
                            EmitElement(OCornerTLGO, coordinate, false, SetId(6,
    isSetId), isSetId);
                        }
                        else if (left && !leftTop && !top)
                        {
                            coordinate.x = coordinate.x - 5;
                            EmitElement(ICornerTRGO, coordinate, false, SetId(3,
    isSetId), isSetId);
                        }
                        else if (!left && leftTop && top)
                        {
                            EmitElement(lineBGO, stepSize * dgMap.GetPosition(i, j),
    false, SetId(12, isSetId), isSetId);
                        }
                        else if (!left && leftTop && !top)
                        {
                            coordinate.z = coordinate.z + 5;
                            EmitElement(ICornerBRGO, coordinate, false, SetId(4,
    isSetId), isSetId);
                        }
                        else if (!left && !leftTop && top)
                        {
                            coordinate.x = coordinate.x + 5;
                            EmitElement(ICornerBLGO, coordinate, false, SetId(1,
    isSetId), isSetId);
                        }
                        else if (!left && !leftTop && !top)
                        {

                        }
                    }
                }
            }
        }
```