ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методу випадкової генерації рівнів для гри в жанрі Tower Defense»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Марія ІГНАТОВА
(підпис)

Виконала: здобувачка вищої освіти група ПДМ-64
Марія ІГНАТОВА

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ігнатовій Марії Володимирівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Розробка методу випадкової генерації рівнів для гри в жанрі Tower Defense

керівник кваліфікаційної роботи: Олесь ДІБРІВНИЙ доктор філософії (PhD),

затверджені наказом Державного університету інформаційно-телекомунікаційних технологій «19» жовтня 2023 року №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вхідні дані до кваліфікаційної роботи:
науково-технічна література, вимоги до поведінки штучного інтелекту

4. Аналіз наявних математичних методів та технологій для розробки методу випадкової генерації рівнів для гри в жанрі Tower Defense.
1. Дослідження інформаційних технологій за темою.
2. Реалізація випадкової генерації рівнів для гри в жанрі Tower
3. Defense.
4. Опис проектування системи.
5. Опис використаних технології.

5. Перелік демонстраційного матеріалу (назва основних слайдів)
   1. Титульний слайд
   2. Мета, об’єкт та предмет дослідження
   3. Ключові елементи рівнів для гри в жанрі tower defense
   4. Існуючі методи вирішення задачі створення рівнів для гри в жанрі tower defense
   5. Процес створення рівня в грі жанру tower defense
   6. Порівняльна характеристика методів генерації мапи для рівня
   7. Порівняльна характеристика методів пошуку шляху
   8. Математична модель методу генерації мапи
   9. Алгоритм генерації рівня
   10. Критерії генерації рівню
   11. Приклад генерації
   12. Результати тестування рівнів
   13. Апробація результатів дослідження
   14. Висновки

   6. Дата видачі завдання «19» жовтня 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вивчення теми магістерської роботи	19.10.23-05.11.23	
2	Вивчення літературних джерел	06.11.23-12.11.23	
3	Складання плану роботи	13.11.23-15.11.23	
4	Узгодження плану роботи та списку використаних джерел з науковим керівником	16.11.23-17.11.23	
5	Аналіз існуючих математичних рішень	18.11.23-23.11.23	
6	Дослідження інформаційних технологій	24.11.23-27.11.23	
7	Розробка модифікованої математичної моделі та проведення дослідження	28.11.23-04.12.23	
8	Оформлення роботи: вступ, висновки, реферат	05.12.23-07.12.23	
9	Розробка демонстраційних матеріалів	08.12.23-10.12.23	

Здобувач вищої освіти _____ Марія ІГНАТОВА
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: роботи 75 с., 37 рис., 21 фор., 3 табл. , 25 джерел.

Мета дослідження: підвищення якості створення рівнів в іграх жанру Tower Defense за рахунок використання автоматичної генерації.

Об'єкт дослідження: процес генерації випадкових рівнів в грі жанру Tower Defense.

Предмет дослідження: алгоритми штучного інтелекту для генерації рівнів.

Короткий зміст роботи: результати проведеного наукового дослідження дають зрозуміти, що використані математичні методи, алгоритмічні моделі, та програмні засоби надають можливість вдосконалити процес створення випадкових рівнів для гри жанру Tower Defense.

Практичні значення отриманих результатів полягає в тому, що на основі проведених теоретичних досліджень було розроблено новий метод підвищення якості створення рівнів в іграх жанру Tower Defense. Результати цього дослідження надалі дають змогу пришвидшити роботу алгоритмів штучного інтелекту генерації рівнів.

КЛЮЧОВІ СЛОВА: TOWER DEFENSE, ШУМ ПЕРЛІНА, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ГЕНЕРАЦІЯ РІВНЯ, АЛГОРИТМ A*

ABSTRACT

Text part of the master's qualification work: 75 pages, 37 figures, 21 formulas, 3 tables, 25 references.

Object of research: the process of generating opponent behavior in turn-based strategy games.

Subject of research: artificial intelligence algorithms for game opponents.

Research goal: improving the quality of opponent behavior in turn-based strategy games through the creation of a proprietary method.

Summary of the work: the results of the conducted scientific research indicate that the employed mathematical methods, algorithmic models, and software tools provide the opportunity to enhance the generation of opponent behavior scenarios in turn-based strategy games.

The practical significance of the obtained results lies in the fact that based on the conducted theoretical research, a new method for generating opponent behavior scenarios in turn-based strategy games has been developed. The outcomes of this research further facilitate the optimization of artificial intelligence algorithms for generating game levels.

KEYWORDS: TOWER DEFENSE, PERLIN NOISE, PROCEDURAL GENERATION, LEVEL GENERATION, ALGORITM A*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувачка ІГНАТОВА М.В. до захисту кваліфікаційної роботи

за спеціальністю 121 Інженерія програмного забезпечення

освітньо-професійної програми «Інженерія програмного забезпечення»

на тему: «Розробка методу випадкової генерації рівнів для гри в жанрі Tower Defense».

Кваліфікаційна робота і рецензія додаються.

Директор ННІ ІТ _____ Андрій БОНДАРЧУК
(підпис)

Висновок керівника кваліфікаційної роботи

Магістерська робота, що виконала студентка Ігнатова М.В., відповідає завданню та завершена вчасно. В ході виконання роботи студенткою був розроблений метод генерації випадкових рівнів для гри жанру Tower Defense. Магістром проаналізовано всі наявні методи розробки рівнів та покращено роботу розробленого методу. По даним й результатам виконаної роботи можна зробити висновок, що студент вдало використовує знання, отримані в Державному університеті інформаційно-телекомунікаційних технологій, правильно і чітко формулює завдання, може розв'язувати їх та доводить рішення до кінцевого результату. При вирішенні поставлених завдань приймає грамотні обґрунтовані рішення. При оформленні текстових та графічних матеріалів застосовувались сучасні інформаційні технології і офісні пакети.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувачки Ігнатової М.В. на оцінку «добре» та присвоїти їй кваліфікацію магістр з інженерії програмного забезпечення.

Керівник кваліфікаційної роботи _____ _Олесь ДІБРІВНИЙ__
(підпис))

«____» _____ 202__ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувачка ІГНАТОВА М.В. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедри ІПЗ _____ Ірина ЗАМРІЙ
(підпис)

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувачки вищої освіти Ігнатової Марії Володимирівни

на тему «Розробка методу випадкової генерації рівнів для гри в жанрі Tower Defense »

Актуальність.

Тема дослідження, присвячена розробці методу випадкової генерації рівнів для гри в жанрі Tower Defense, є актуальною в сучасному контексті розвитку ігрової індустрії. Створення ефективних алгоритмів генерації рівнів є ключовим для покращення ігрового досвіду та забезпечення нових вражень гравцям у цьому жанрі.

Позитивні сторони.

1. Робота виявляє глибоке розуміння основних принципів генерації рівнів у жанрі Tower Defense та засобів, які використовуються для підвищення якості геймплею.
2. Здобувачка продемонструвала високий рівень технічних знань і навичок, використовуючи методи шумів Перліна та алгоритм A*, щоб створити модель генерації рівнів для гри в жанрі Tower Defense.

Недоліки.

1. У деяких випадках може бути корисніше додатково проаналізувати результати тестування створеного методу та його ефективність у різних умовах гри.
2.Потрібно більше деталей щодо можливих обмежень створеної моделі генерації рівнів, особливо з урахуванням варіативності вимог до складності рівнів у Tower Defense.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна робота на здобуття ступеня магістра заслуговує оцінку "добре", а здобувачка* ***Ігнатова Марія Володимирівна*** *заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення.*

Рецензент: _____ _____
науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

TD – Tower Defense

ГА – Генетичний алгоритм

ACO - Техніка оптимізації колонії мурах

ВСТУП

Сучасна ігрова індустрія постійно еволюціонує, а нові технології та концепції надають можливості покращувати якість ігрового досвіду. Один з жанрів, що користується великим попитом, це Tower Defense - стиль ігор, що вимагає від гравця захищати свою територію від хвиль ворогів, будуючи й розміщуючи вежі та інші об'єкти для нанесення шкоди нападаючим.

Те, що робить Tower Defense настільки захоплюючим, це його стратегічний аспект. Гравець повинен ретельно планувати та розташовувати оборонні споруди так, щоб оптимально управляти ресурсами та відбити атаки противників. Цей жанр ігор не тільки пропонує захоплюючі битви, а й вимагає логічного мислення та стратегічної обачності від гравця.

У зв'язку зі зростаючим інтересом до Tower Defense, індустрія постійно шукає нові шляхи для покращення цього жанру. Одним з ключових аспектів є розробка нових методів генерації рівнів, які дозволяють створювати рівні швидше та якісніше.

Завдання роботи полягає в таких пунктах:

1. Аналіз існуючих методів та підходів для генерації випадкового рівня ігор жанру Tower Defense.
2. Створення власного алгоритму генерації випадкового рівня ігор жанру Tower Defense.
3. Впровадження алгоритму в програмну розробку ігрової моделі.
4. Проведення тестування розробленого методу.
5. Порівняння отриманих результатів з вже існуючими методами.

Мета дослідження: підвищення якості створення рівнів в іграх жанру Tower Defense за рахунок використання автоматичної генерації.

Об’єкт дослідження: процес генерації випадкових рівнів в грі жанру Tower Defense.

Предмет дослідження: алгоритми штучного інтелекту для генерації рівнів.

Результати проведеного наукового дослідження дають зрозуміти, що використані математичні методи, алгоритмічні моделі, та програмні засоби надають можливість вдосконалити процес створення випадкових рівнів для гри жанру Tower Defense.

Практичне значення отриманих результатів полягає в тому, що на основі проведених теоретичних досліджень було розроблено новий метод підвищення якості створення рівнів в іграх жанру Tower Defense. Результати цього дослідження надалі дають змогу пришвидшити роботу алгоритмів штучного інтелекту генерації рівнів.

Актуальність даної розробки полягає в необхідності підвищення якості та різноманітності генерації випадкових рівнів в іграх жанру Tower Defense. Однією з важливих складових успіху гри в цьому жанрі є створення цікавих та різноманітних рівнів, що дозволяє гравцям отримати непередбачуваний та захоплюючий досвід.

Наукова новизна розробленого методу для генерації випадкових рівнів в іграх жанру Tower Defense полягає в інтеграції двох ключових аспектів: використання методу шуму Перліна для генерації рельєфної мапи та застосування алгоритму на основі A* для оптимального прокладання шляху в грі. Цей підхід дозволяє створювати рівні, які не лише відрізняються від попередніх за своєю унікальністю та різноманітністю, а й забезпечують оптимальну розбудову географії рівня та визначення оптимального маршруту для ворожих атакуючих персонажів у грі.

Додатково, розроблений метод використовує алгоритми штучного інтелекту для автоматичного створення географічно різноманітних рівнів, що відповідають потребам балансу гри та забезпечують цікаву взаємодію між гравцем та ворожими силами. Комбінація використаних методів дозволяє покращити процес генерації рівнів, забезпечуючи плавну та наочну ігрову динаміку, що робить цей підхід інноваційним у відношенні до попередніх методів генерації в даному жанрі.

1 СТВОРЕННЯ ТА МОДЕЛЮВАННЯ РІВНІВ ДЛЯ ГРИ В ЖАНРІ TOWER DEFENSE

1.1 Формулювання поняття жанру та особливостей створення рівнів для гри Tower Defense

Жанр ігор Tower Defense (TD) став предметом великого інтересу для дослідників в галузі інтерактивних розваг. У цьому контексті необхідно проаналізувати та систематизувати основні характеристики цього жанру, а також виокремити ключові аспекти створення рівнів для ігор даного типу.

Жанр Tower Defense розглядається як під-жанр стратегічних відеоігор. Важливою стратегією в цьому жанрі є вибір та розміщення оборонних елементів. Загалом культові ігри Tower Defense мають однакову основну концепцію: захищати точку, будуючи споруди для відштовхування хвиль ворожих нападів. Після побудови споруди автоматично спрямовуються на ворогів та атакують їх. Кожна "вежа" діє по-різному: деякі атакують один об'єкт, інші - кілька цілей, а деякі навіть не атакують, а лише надають підтримку іншим вежам. Гравець вирішує, які споруди будувати та де розміщувати їх. Задача ігор у жанрі Tower Defense полягає у можливості гравця зупиняти чи сповільнювати просування ворожих сил шляхом розташування веж, що надають можливості для нападу на полі бою. Деякі класичні ігри у жанрі Tower Defense мають схожість із іграми у жанрі рольових ігор (RPG), оскільки вони мають геймплей на базі чергування ходів. Є

У найпростіших варіантах жанру та прагнуть дістатися кінця карти, що представляє ігрове поле. У найпростішому випадку карта починається з правої сторони екрана і закінчується на лівій. Гравець втрачає життя, якщо противник дістається кінця карти, але отримує нагороду, якщо його знищує одна з веж. Гравець впливає на гру, купуючи, продаючи, розміщуючи та покращуючи вежі. Під час розвитку гри, противники стають сильнішими, і гравець спонукається до подальшого заповнення карти вежами.

Рис. 1.1. Приклади ігор в жанрі Tower Defense

Ігри у жанрі TD створюють для гравця завдання, які потрібно подолати. Зазвичай ці ігри надають ворогові можливість пересуватися по кількох шляхах, оборону яких має забезпечити гравець. У деяких іграх у жанрі Tower Defense або режимах гри кількість шляхів збільшується з розвитком гри. Дизайнери ігор створюють для своїх ігор різноманітні типи та варіації ворогів. Вони атакують в групах по декілька осіб, хоча їх характеристики змінюються від гри до гри. Хвилі ворогів часто з'являються, ускладнюючи ситуацію. Час від часу відбувається бій з босом, який потрібно подолати. Разом із зростанням рівня ворогів також покращуються вежі. Нові вежі стають потужнішими і мають різні варіанти зброї, що додає можливості, такі як дальність стрільби, швидкість, пошкодження від удару та інші, в залежності від гри. Також з'являються інші спеціальні можливості. Популярні ігри у жанрі TD залучають гравця, роблячи розміщення веж критичним для прогресу. Стратегія оборони гравця стає дуже важливою для просування в грі. Ігри у жанрі Tower Defense можна знайти на мобільних пристроях, таких як App Store для iPhone та Google Play для Android-телефонів, на консолях, у веб-браузерах і на платформах, таких як Steam.

Tower Defense виник під час епохи Flash-ігор та досі є популярною та простою стратегічною грою. У найпростішому випадку карта починається з правої сторони екрана і закінчується на лівій. Гравець втрачає життя, якщо противник

дістається кінця карти, але отримує нагороду, якщо його знищує одна з веж. Гравець впливає на гру, купуючи, продаючи, розміщуючи та покращуючи вежі. Незважаючи на багато-форматність жанру, у кожній грі Tower Defense можна виокремити такі основні елементи[1]:

- карта,
- шлях,
- вежі,
- противники,
- система нагород.

Ігри жанру Tower Defense представляють собою унікальну форму інтерактивного досвіду, базованого на стратегічному плануванні та тактичному мисленні гравця. Розуміння основних принципів цього жанру і особливостей створення рівнів для таких ігор дозволяє розширити можливості розвитку цього напрямку в індустрії відеоігор.

Також для розуміння сутності жанру розглянемо деяких представників жанру:

- Kingdom Rush;
- Plants Vs. Zombies;
- Defense Grid 2.

"Kingdom Rush" - це комп'ютерна гра в жанрі "Tower Defense", яку створила студія Ironhide Game Studio та опублікувала Armor Games у 2011 році. Гра отримала позитивні відгуки критиків і геймерів, вважаючи серію "Kingdom Rush" однією з кращих у своєму жанрі.

Сюжет розгортається у фантазійному світі. Уздовж передбаченого шляху руху супротивників існують пусті слоти, де гравець може встановлювати башти. У грі доступні чотири типи захисних споруд: магів, лучників, казарм та артилерії, а також два види заклинань: виклик допомоги та вогняний дощ. На початку кожного рівня у гравця є обмежена сума грошей для придбання перших башт. Основна мета

полягає в знищенні всіх ворогів до досягнення ними кінця шляху, відміченого синім щитом. За вбивство ворогів гравець отримує грошову винагороду, яку можна використовувати на покращення наявних захисних споруд чи побудову нових. Гра містить 26 рівнів і 48 видів ворогів. Після успішного завершення рівня гравець отримує зірки, які можна використовувати для купівлі удосконалень для башт та заклинань. Крім основного режиму гри, доступні ще два варіанти: "Героїчний" та "Залізний виклик", що відрізняються більшою складністю.

Рис. 1.2. рівень в Kingdom Rush

"Plants vs. Zombies" - це франшиза відеоігор, розроблена компанією PopCap Games, що належить до Electronic Arts. Серія слідує пригодам Девіда "Божевільного Дейва" Блейзінга, який використовує рослини для захисту від нападу зомбі, очолюваного доктором Едгаром Джорджем Зомбоссом. Перша гра, Plants vs. Zombies (2009), була розроблена і випущена PopCap перед її придбанням EA. Після придбання PopCap Games EA розширила гру в франшизу з виданнями на різних платформах.

Гра "Plants vs. Zombies", розроблена PopCap Games, представляє собою видовищну гру в жанрі Tower Defense, випущену у 2009 році. У цій грі гравець керує будинком, який обороняється від атак зомбі за допомогою різноманітних рослин, керуючи власними стратегічними вміннями та логікою.

Головний персонаж, Божевільний Дейв, взявши на себе роль захисника будинку, використовує рослини для стримування та відбиття вторгнення зомбі.

Гравець розміщує рослини на стратегічно важливих позиціях, кожна з яких має свої унікальні властивості для боротьби з різними видами зомбі. Розгортання рослин та їх оптимальне використання дозволяють гравцеві захистити свій будинок від наступу зомбі, використовуючи стратегічні рішення та інтелектуальні здібності.

Гра пропонує багато етапів зростаючої складності, кожен з яких вимагає від гравця використання стратегічного мислення та інноваційних підходів для досягнення успішного вирішення завдань. Вдалий баланс екшну та стратегії робить "Plants vs. Zombies" захоплюючою для гравців різного віку та ігрового досвіду. Непередбачуваність розвитку ситуацій та необхідність вдумливих рішень роблять цю гру незабутньою для шанувальників стратегічних ігор.

Рис. 1.3. Рівень в грі Plants vs. Zombies

Defense Grid 2 (DG2) є продовженням гри Defense Grid: The Awakening від Hidden Path Entertainment 2008 року. Встановлюючи планку як гра нового рівня якості у жанрі Tower Defense, Defense Grid 2 вводить нові світи та загрози, щоб випробувати стратегії гравця в розташуванні веж.

Завдяки новому вигляду, захоплюючій одиночній кампанії та додаванню нових режимів гри, таких як онлайн битва гравців один на одного та багатокористувацький кооператив, кожне проходження відкриває нові можливості та виклики для поціновувачів жанру, що безумовно робить гру цілковито

конкурентною на цільовому ринку навіть і сьогодні.

DG2 має сюжетну кампанію з 21 захопливою мапою, динамічним рухом рівнів, розширеною історією та групою персонажів, сотнями викликів у режимі складності, новими мультиплеерними режимами, процедурно створеною аудіо-супроводженням та можливістю гравців будувати власні рівні.

DG2 також представляє гравцям DG Architect на ПК - інструмент для створення рівнів, пов'язаний з Steam Workshop, де гравці можуть створювати унікальні рівні для спільного використання та, можливо, навіть продажу у разі вибору. Що є унікальним елементом для ігор цього жанру, але не вирішує питання зі створенням рівнів загалом.

Рис. 1.4. Рівень в грі Defense Grid 2

DG2 для ПК була частково фінансована Dracogen, приватним інвестором після кампанії Kickstarter 2012 року, яка фінансувала розширення оригінальної гри - Defense Grid: Containment.

Отже ігри з вдалим вирішенням створення рівнів та наповнення в жанрі Tower Defense мають свою увагу поза часу, що робить питання створення якісної продукції в цьому направлені більш необхідним.

Також важливим є аналіз того, зо є важливим для потенційної майбутньої авдиторії. За результатами загальних опитувань великої кількості гравців[3]:

- Стать впливає на сприйняття гри: Чоловіки в цілому оцінювали гру краще у компонентах основного геймплею та соціальної присутності, за винятком після гри.
- Досвід гравця в інших іграх впливає на сприйняття: Учасники, які раніше грали в інші ігри Tower Defense, надали вищі оцінки у компонентах основної ігрової механіки, соціальної присутності та після гри порівняно з учасниками, які не грали раніше.
- Необхідність у вдосконаленні гри перед наступним етапом тестування: Загальний бал, свідчить про те, що для подальшого розвитку гри потрібні покращення.
- Низькі оцінки були пов'язані з недостатнім рівнем виклику та різноманітності в грі. Тести підкреслили необхідність більш варіативних та складніших ворогів, а не складніших систем управління вежами.

Що свідчить, що створення генерації рівнів для цього жанру комп’ютерних ігор покращить досвід використання подібного програмного продукту.

З постійно зростаючим вибухом нових відеоігор і новими розробками в ігровому світі питання про критику відеоігор стають складнішими. Різні визначення, які гравці та критики використовують, щоб вирішити, що таке гра та що робить гру успішною, часто призводять до різних уявлень про те, як ігри досягають успіху чи провалу. Створення рівнів для ігор є одним з найважливіших аспектів оцінки якості гри в цілому. Відповідно вміння створити різноманітні і врівноважено складні рівні для ігор в жанрі Tower Defense являє собою головну задачу в розробці майбутнього програмного продукту[4].

Окремо, спеціально для розробляємого методу було проведено для аналізу необхідності та актуальності розробки соціальне опитування фокус групи, щодо розробленого методу та зацікавленості в ньому потенційної авдиторії віком від 10 до 40 років.

Рис. 1.5. Опитування потенційної авдиторії

Як можна побачити з результатів опитування більшість, а саме 91,3 відсотка опитаних, раніше грали в гру жанру Tower Defense, що говорить про те, що даний жанр є досить популярним серед вибраної цільової авдиторії.

Також в опитуванні було зазначено уточнення з приводу того, чи були б нові ігри подібного жанру цікаві потенційним гравцям. В опитування шкала 5 означала – зацікавлені, а 1 – не зацікавлені, відповідно.

За проведеними аналітичними підрахунками результатів опитування було виявлено, що більшість, а саме 80% опитаних були б не проти використовувати нові версії представників цього жанру. 9.6% вирішили дати посередню відповідь і тільки 10.4% обрали негативну відповідь.

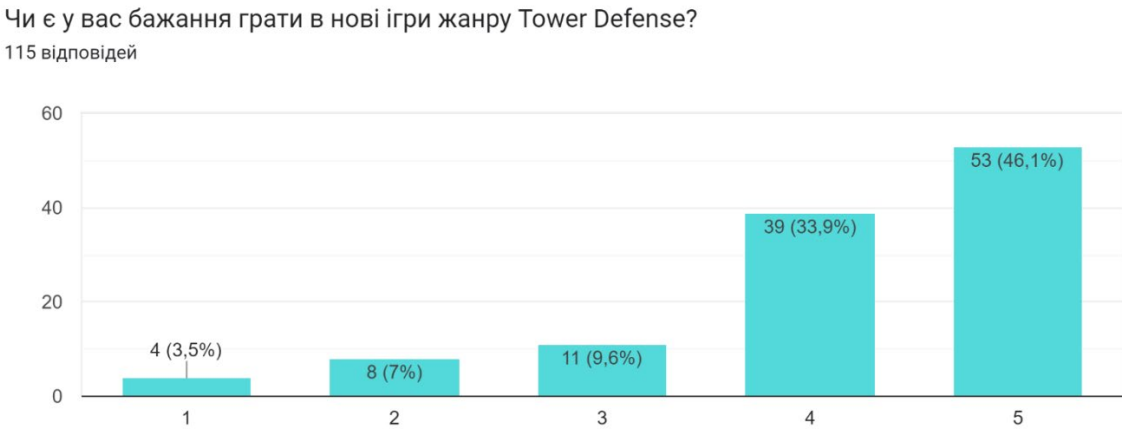


Рис. 1.6. Статистика відповідей на питання актуальності жанру гри

Загалом, можна зробити висновок з проаналізованої інформації, що згідно статистичних даних здебільшого авдиторія зацікавлена в нових розробках в цьому направлені створюваних ігор. Сам жанр досі є популярним і актуально створювати продукцію, що може конкурувати на ринку. І як було зазначено вище, створення рівнів в даному направлені ігрових жанрів є одним з ключових інструментів цього питання.

1.2 Аналіз наукових та практичних підходів до створення рівнів в іграх

Отже Tower Defense — це популярний під-жанр стратегії в реальному часі, який потребує детального проектування рівнів і балансування складності, щоб створити приємний досвід гравця. Досі активним та найлегшим методом створення ігор жанру Tower Defense можна вважати створення подібних рівнів власноруч. Створення рівня в грі жанру Tower Defense вимагає послідовного підходу, починаючи з визначення концепції. Спочатку визначається загальна ідея рівня, включаючи ворогів, їхні шляхи пересування до цілі, області для розміщення веж і ресурси. Також створюється карта рівня, визначаються на ній більш конкретно шляхи для руху ворогів та розміщення пунктів ресурсів і веж для оборони. Важливо також визначити типи ворогів та їх характеристики, такі як швидкість, здоров'я, сила атаки. Наступний етап - тестування та налагодження рівня, під час якого виправляються можливі недоліки, щоб забезпечити баланс і цікавість. Завершальним етапом є остаточне доведення рівня до повністю функціонального для виправлення можливих проблем та переконання у готовності рівня для гри. Такий послідовний підхід допомагає створити рівень у грі Tower Defense, забезпечуючи якість та цікаві ігрові можливості, але має свої недоліки, які ми розглянемо детально.

Головним недоліком використання для цього методу - є час. Цього ресурсу на виконання кожного етапу для всіх рівнів окремо знадобиться набагато більша кількість ніж з генерацією рівня. До того ж власноручне створення рівнів може призвести до повторюваних або передбачуваних елементів, що зробить ігровий

процес менш різноманітним та цікавим для гравця.

Окрім часу та повторюваності головною проблемою створення рівня власноруч є використання більшої кількості персоналу та їх можливостей, які можна було б спрямувати в інші частини проєкту, тим самим покращивши результат.

Загальний алгоритм створення рівня Tower Defense буде виглядати наступним чином:

Рис. 1.7. Алгоритм створення рівня в грі Tower Defense

Відповідно, для автоматизації створюваного рівня і прискорення його створення можна деякі етапи алгоритму зробити генерованими та автоматичними. Процедурне генерування контенту вже застосовується у багатьох популярних іграх, таких як Spelunky та No Man's Sky, і прогнозується, що з розвитком складних технік штучного інтелекту ми побачимо з'явлення абсолютно нових типів ігор, які

були б неможливі без використання подібних технологій. Наводиться приклад гри AI Dungeon 2, яка використовує мовну модель GPT-2 від OpenAI.[5]

Отже, для покращення процесу створення рівнів в грі Tower Defense пропонується зробити автоматичну генерацію мапи та пошук шляху. Що вирішить проблему з використанням більшої кількості часу на створення нового рівня.

Підводячи підсумки, у контексті гри Tower Defense, автоматизована генерація мапи та пошуку шляху дозволить розробникам швидше створювати різноманітні та цікаві рівні, забезпечуючи більше можливостей для гравців. Це зменшить час, витрачений на ручне створення рівнів, та відкриє шлях до інноваційних ігрових досвідів, які були б неможливі без використання передових технологій генерації контенту. Продовжуючи цей шлях розвитку, можна очікувати появу нових та унікальних ігор, які будуть привабливі для широкої аудиторії гравців.

1.3 Методи створення рівнів для ігор жанру Tower Defense

Створення базової структури мапи з використанням алгоритмів генерації можливо за допомогою такого ряду методів:

- Метод відхилення середини;
- Метод генерації за допомогою клітинного автомату;
- Алгоритм «Квадрат-ромб»;
- Метод шуму Перліна;
- Метод шуму Сімплекса.

Метод відхилення середини: Простий алгоритм полягає у створенні кількох випадкових синусоїдальних функцій для керування висотою пагорба. Він працює, починаючи з випадкових значень у двох точках, обчислюючи їх середнє значення в середині відрізка, додаючи деякий випадковий шум, а потім рекурсивно повторює цей процес для середини між вихідними точками та центральною точкою. Алгоритм базується на принципі використання прямолінійних відрізків для

створення візуально реалістичних рельєфів. Починаючи з вихідного відрізка, метод обчислює середину цього відрізка та відхиляє її на випадкове значення. Це відхилення може бути застосоване у напрямку, перпендикулярному до відрізка, або лише до координати у середині. Кожна ітерація робиться на основі попередньої, розбиваючи відрізок на два менших відрізки, для яких знову обчислюються середини та відбувається відхилення. Цей процес може повторюватися ітеративно або рекурсивно доти, поки відрізки не досягнуть мінімальної ширини (наприклад, два пікселі). Такий підхід дозволяє ефективно моделювати природні форми рельєфів у візуальних додатках, забезпечуючи при цьому зручність і гнучкість управління структурою та деталізацією створеного ландшафту. Рекурсивна природа алгоритму дозволяє легко створювати деталізовані пагорби або інші елементи ландшафту на різних рівнях масштабування.

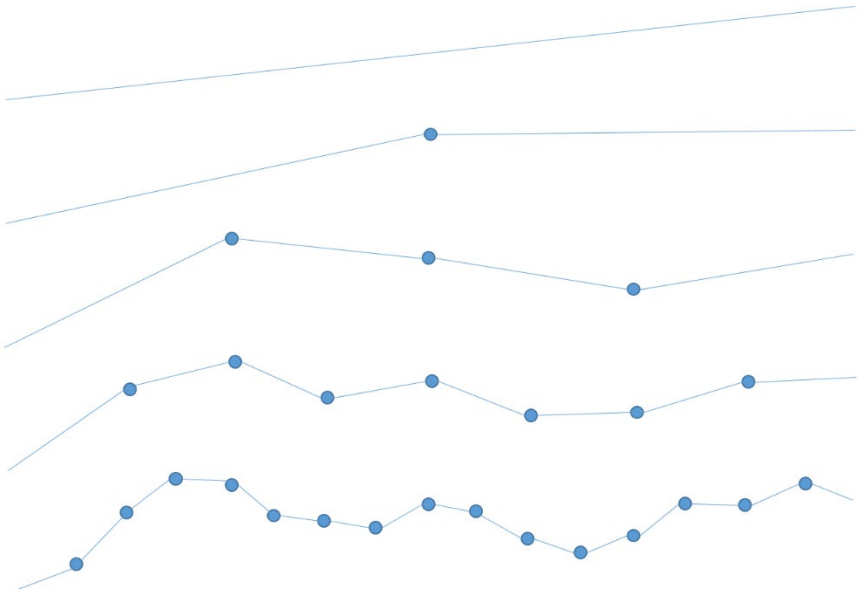


Рис. 1.8. Принцип роботи методу відхилення середини

Оскільки цей метод базується на відхиленні від прямої лінії, результатом можуть бути вибіги або виступи, які не завжди підходять для генерації ландшафту в грі Tower Defense, де може бути важливо забезпечити певну структуру або функціональність для об'єктів на рівні. До того ж метод відхилення середини може вимагати значних ресурсів та часу для створення складних або великих рівнів. Це

може стати проблемою, особливо при необхідності швидкої генерації нових рівнів для ігрового процесу.

Підхід на основі клітинного автомату базується на ідеї початкової випадкової сітки з 0 (стіни) та 1 (проходи). Потім застосовують простий набір правил до кожної комірки, щоб отримати нову сітку, повторюючи процес кілька разів. Така генерація є моделями обчислень, що ґрунтуються на принципі роботи зі станами клітинок у визначеній сітці. Вони варіюються за типом сітки, кількістю можливих кольорів (або станів) та сусідством клітинок. Для типу сітки визначається форма для обчислень: одновимірна лінія, двовимірні квадратна, трикутна, шестикутна сітки або решітка Картезіана у довільній кількості вимірів.. Найпростішим вибором кількості можливих станів клітинки у сітці є два стани (бінарний автомат), де 0 зазвичай відповідає "білому", а 1 - "чорному", але також можна розглядати інші числові значення. "Найближчі сусіди" - найпростіший вибір, де тільки безпосередньо сусідні клітинки впливають одна на одну. У двовимірних автоматах на квадратній сітці використовують "квадрат Мура" та "сусідство фон Неймана" (ромбоподібний сусід).

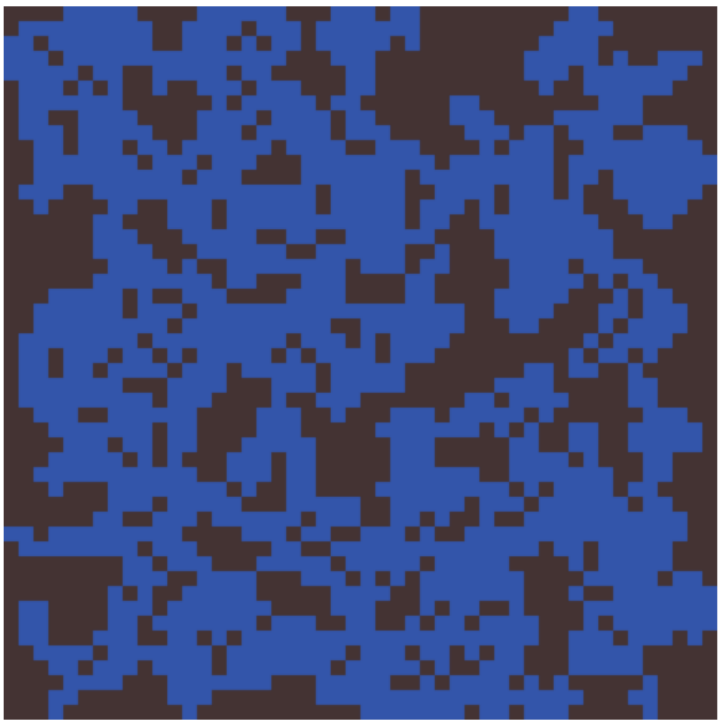


Рис. 1.9. Приклад роботи генерації рівня на основі методу клітинного автомату

Алгоритм клітинного автомату є відносно простим у реалізації. Його легко реалізувати з використанням масивів чи сіток клітин, зменшуючи складність коду та вимоги до ресурсів. Не дивлячись на зручність даного підходу є й проблемні сторони. Певні конфігурації клітинних автоматів можуть призводити до нерівномірності або недосконалості генерованого ландшафту, що може створювати складнощі в створенні рівномірно зваженого контенту гри.

Алгоритм "Квадрат-ромб": Цей алгоритм генерує рельєф (тобто пагорби) за допомогою фрактального підходу. Він починається з деяких випадкових значень, що розташовані заздалегідь, а потім обчислює шумові значення між ними, обчислюючи середні та додаючи деякий шум. Цей процес повторюється багато разів, доки кожна комірка не отримає своє шумове значення.

Алгоритм розпочинається з масиву, який утворює квадратну сітку розміром 2n + 1, де n - це ступінь двійки. Чотири початкові точки на кутах цього масиву спочатку заповнюються відповідними початковими значеннями. Потім виконуються чергові кроки - ромбові та квадратні - доки не будуть визначені всі значення у масиві. Для кожного квадрата у масиві встановлюється середня точка як середнє значення чотирьох кутових точок, додаючи до цього випадкове значення. Потім для кожного ромба у масиві встановлюється середня точка як середнє значення чотирьох кутових точок.

Кожне випадкове значення множиться на константу масштабу, яка зменшується з кожною ітерацією на множник 2−h, де h - значення між 0.0 та 1.0 (менші значення створюють бурхливіший ландшафт).

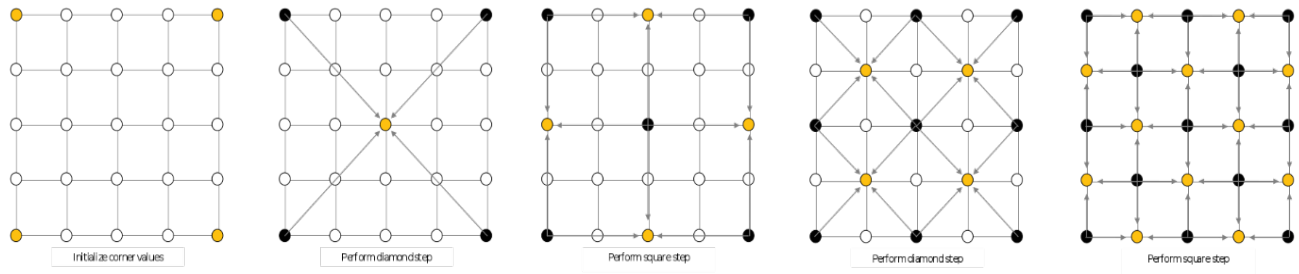


Рис. 1.10. Алгоритм "Квадрат-ромб"

Під час ромбових кроків точки, що розташовані на краях масиву, матимуть лише три встановлені сусідні значення замість чотирьох. Існують кілька підходів для вирішення цієї складності, простий спосіб - враховувати середнє значення лише трьох сусідніх точок. Ще один варіант, коли використовується четверте значення з протилежного краю масиву. При використанні постійних початкових значень кутів, цей метод також дозволяє плавно зшивати створені фрактали, уникнувши розривів в зображенні.

Стандартний підхід генерації шуму Перліна, який полягає в тому, щоб відобразити вивід функції шуму Перліна на конкретні плитки. Функція шуму Перліна формується шляхом наложення кількох карт шуму з різною амплітудою і частотою, що називаються "октавами". Шум Перліна - це складний алгоритм для генерації випадкового рельєфу, який залишається гладким та безперервним. Найкраще в тому, що цей алгоритм працює добре з процедурною генерацією, що означає, що він може створювати частину світу, а потім, рухаючись до краю, можна генерувати більше рельєфу, який буде безперервно з'єднуватися.

Формула для обчислення інтерполяції шуму Перліна виглядає наступним чином:

$$\mathrm{P} = (\mathrm{x}, \mathrm{y}), i = \mathrm{floor}(\mathrm{x}), \mathrm{j} = \mathrm{floor}(\mathrm{y}) \quad (1.1)$$

$$g_{00} = \text{gradient at}(\mathrm{i},), g_{10} = \text{gradient at } (\mathrm{i}+1,)\,, \quad (1.2)$$

$$g_{01} = \text{gradient at}(\mathrm{i}, +1), g_{11} = \text{gradient at } (\mathrm{i}+1, +1)\,, \quad (1.3)$$

$$u = x - i, v = y - j, \quad (1.4)$$

$$n_{00} = g_{00} \cdot \begin{bmatrix} \mathrm{u} \\ \mathrm{v} \end{bmatrix}, n_{10} = g_{10} \cdot \begin{bmatrix} \mathrm{u}-1 \\ \mathrm{v} \end{bmatrix}, n_{01} = g_{01} \cdot \begin{bmatrix} \mathrm{u} \\ \mathrm{v}-1 \end{bmatrix}, n_{11} = g_{11} \cdot \begin{bmatrix} \mathrm{u}-1 \\ \mathrm{v}-1 \end{bmatrix}, \quad (1.5)$$

$$n_{x0} = n_{00}\big(1 - f(u)\big) + n_{10} f(u), n_{x0} = n_{01}\big(1 - f(u)\big) + n_{11} f(u)\,, \quad (1.6)$$

$$n_{xy} = n_{x0}\big(1 - f(u)\big) + n_{x1} f(v) \quad (1.7)$$

Де перша формула визначає внутрішню частину шуму Перліна для точки (x, y).

Формули 1.2, 1.3 і 1.4 обчислюють градієнти (вектори напрямку) для кожного з чотирьох кутів, що оточують цю точку.

Формули 1.5 обчислюють внутрішні добутки градієнтів та відносні різниці між поточними координатами (x, y) та нижчими цілими значеннями цих координат (зазвичай це (x, y), (x+1, y), (x, y+1), (x+1, y+1)).

Формули 1.6 використовують обчислені внутрішні добутки та різниці для інтерполяції значення шуму Перліна між чотирма кутами.

Формула 1.7 об'єднує значення інтерполяції між кутами для отримання кінцевого значення шуму Перліна для точки (x, y).

Ці формули використовуються для обчислення значень шуму Перліна між різними точками, використовуючи лінійну інтерполяцію між градієнтами у просторі.

Шум Сімплекса - це варіант шуму Перліна, який використовує трикутники замість квадратної сітки для розміщення векторів градієнта. Його краще використовувати для додаткових функцій, оскільки його дуже складно реалізувати.

Другим етапом алгоритму створення рівня в грі Tower Defense є створення шляхів ворогів. Оскільки карта буде генеруватись самостійно – відповідно необхідно визначити алгоритм пошуку шляху, для того, щоб цей етап створення рівня також відбувався автоматично.

Для автоматичного пошуку шляху існує декілька най поширеніших методів з них найкращими можна вважати такі як:

- Алгоритм A*;
- Алгоритм Дейкстри;
- Генетичний алгоритм
- Техніка оптимізації колонії мурах (ACO).

Алгоритм A* представляє собою ефективний метод пошуку найкоротшого шляху між двома вузлами у графі. Його використовують у багатьох жанрах комп’ютерних ігор, таких як стратегії в реальному часі, tower defense, рольові ігри та ігри з гонками. Алгоритм A* вирішує проблему пошуку шляху в різних контекстах, включаючи навігацію не грабельних персонажів у відеоіграх для досягнення певних цілей.

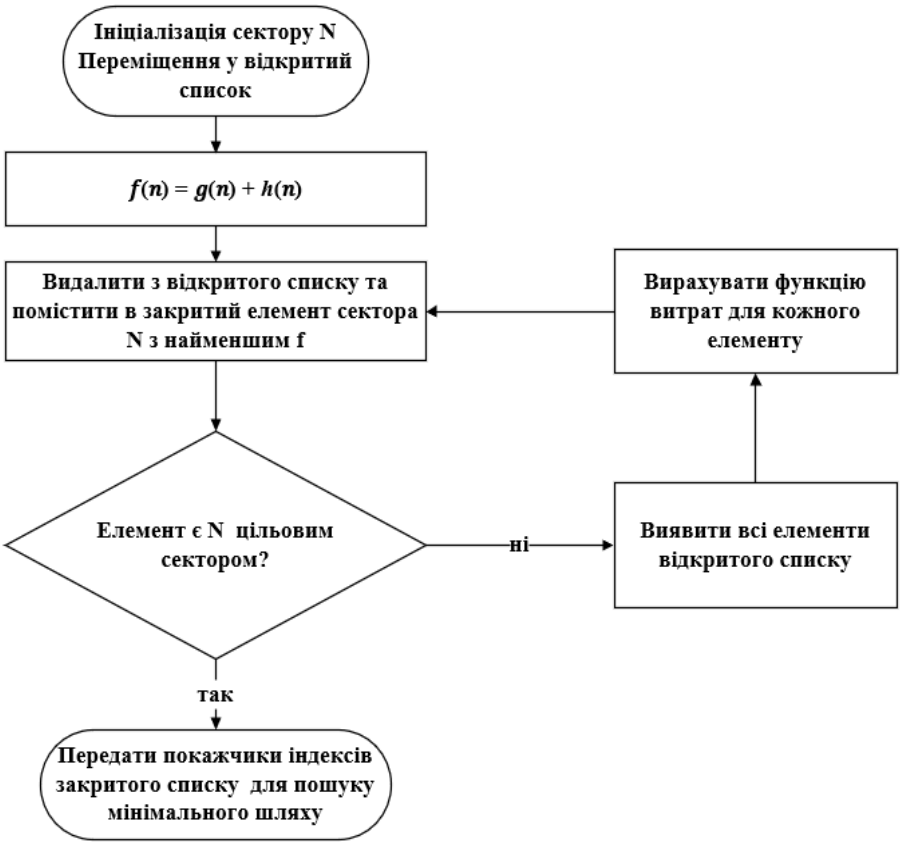


Рис. 1.11. Алгоритм A*

Основна ідея алгоритму полягає у виявленні шляху за рахунок послідовного пошуку мінімального та найдешевшого його варіантів. Для оцінки кращого маршруту A* використовує евристичну функцію, яка дозволяє алгоритму оцінити потрібний шлях швидко та точно. Відмінність алгоритму A* полягає в його простоті в розумінні та логіці виконання. Ця простота робить його популярним серед програмістів, оскільки він дозволяє знайти найкоротший шлях у графі, використовуючи евристичний підхід. Функція $f(n)$, яка використовується для визначення вузла, обчислюється за формулою:

$$f(n) = g(n) + h(n), \quad (1.8)$$

Де $g(n)$ представляє вартість, необхідну для досягнення цільового вузла від вихідного вузла. $g(n)$ обчислює вартість досягнення цільового вузла на даний

момент. $h(n)$ представляє евристичне значення, оцінку від вузла до цільового вузла. Якщо у сітці є перешкоди, $f(n)$ оцінює та обирає найнижчу вартість для отримання хорошого результату. Алгоритм A* стає алгоритмом Дейкстри, коли $h(n)$ дорівнює нулю, що гарантує знаходження найкоротшого шляху.

Починаючи зі специфіки роботи A* з урахуванням вартостей і евристичних оцінок для знаходження найбільш оптимального шляху, ми переходимо до алгоритму Дейкстри, який фокусується на знаходженні найкоротшого шляху між визначеними вузлами, розглядаючи відстані від стартового вузла до всіх інших. Обидва ці алгоритми працюють у графах, але з відмінними підходами до вирішення задачі знаходження шляху.

Алгоритм Дейкстри допомагає знайти найкоротший шлях між двома визначеними вузлами. Починаючи з вузла A як стартового, алгоритм вибирає шлях з найменшою відомою відстанню. Переходячи на вузол з найменшою відстанню, він перевіряє кожен з його сусідніх вузлів. Для кожного сусіднього вузла алгоритм обчислює відстань від стартового вузла. Якщо обчислена відстань або поточний шлях менші за інші, то алгоритм оновлює найкоротший шлях для цього вузла.

Алгоритм Дейкстри, хоча і ефективний у знаходженні найкоротшого шляху між двома вузлами, має певні обмеження. Однією з основних характеристик алгоритму Дейкстри є те, що він працює лише з невід'ємними вагами ребер, тобто з ребрами, де відстань або вага не може бути від'ємною. Крім того, алгоритм Дейкстри не може оптимально працювати у графах з вагами, які можуть змінюватися. Такі обмеження роблять його менш універсальним у порівнянні з більш узагальненими алгоритмами, такими як A*, що використовують евристичні значення для більш швидкого і точного знаходження шляху в різних умовах.

Крім того, алгоритм A* має більшу гнучкість, оскільки він може працювати з різними типами графів і враховувати різні умови, включаючи невизначені ваги ребер та змінні умови шляху. Використання комбінації функції оцінки та евристичних значень робить алгоритм A* більш адаптивним до різних умов та дає йому перевагу у багатьох сценаріях пошуку шляху порівняно з алгоритмом Дейкстри.

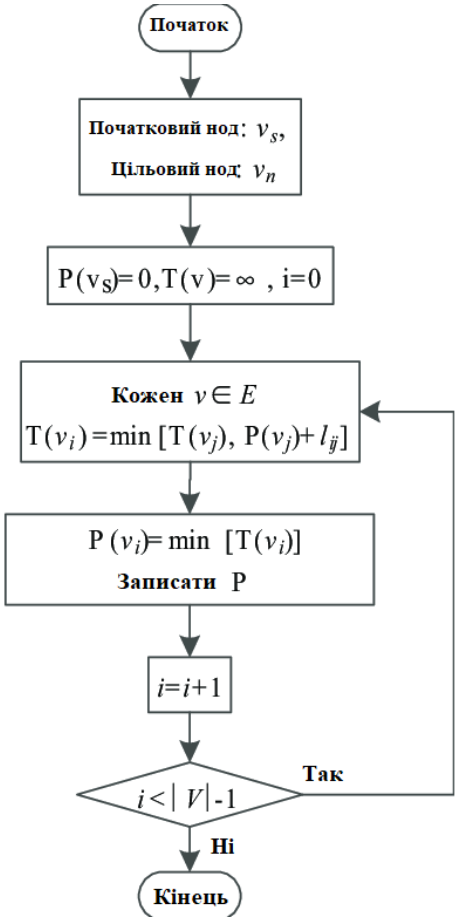


Рис. 1.12. Алгоритм Дейкстри

Науковий метод, що застосовується в комп'ютерній науці, — генетичний алгоритм, став одним з популярних підходів до пошуку рішень. Він застосовується для отримання евристичних рішень у великому просторі оптимізації та пошуку. Генетичний алгоритм був представлений Джоном Голландом. Генетичний алгоритм використовує біологічні методи, такі як мутація та успадкування. ГА перетворює змінні прийняття рішення пошукової задачі у рядки. Рядки діють як кандидатські рішення, і сама пошукова проблема називається хромосомами. Алфавіт виступає як гени, а значення генів називають алелями. Наприклад, у випадку проблеми подорожуючого продавця маршрут представляє хромосоми, а місто — ген. В комп'ютерних іграх генетичний алгоритм застосовується як алгоритм пошуку шляху. Генетичний алгоритм разом з пошуком найкращого шляху використовували для оптимізації пошуку шляху між двома точками.

Результати показали, що ГА демонструє кращу продуктивність на мапах з перешкодами порівняно з пошуком найкращого шляху. Крім того, експерименти щодо оптимізації пошуку шляху з використанням генетичного алгоритму разом з алгоритмом A* показали поліпшення часу пошуку, особливо на мапах з перешкодами. Інший висновок показав, що генетичний алгоритм виявляє шляхи швидше, ніж алгоритм A* за відведений час.

Рис. 1.13. Генетичний алгоритм

Техніка оптимізації колонії мурах є методом, що надихнувся поведінкою мурах під час пошуку шляху від їх мурашника до джерела їжі. Цей метод ґрунтується на здатності мурах знаходити найкоротший шлях між мурашником та джерелами їжі. Головна мета Ant Colony Optimization - знаходження найменш вартісного шляху у графі. ACO використовується в різних сферах, включаючи відеоігри, робототехніку, підводні човни та інші. Техніка ACO володіє перевагою над алгоритмом A* - вона адаптується до змін у реальному часі і може пристосовуватись до нових відстаней. Багато дослідників застосовують і

вдосконалюють метод ACO для розв'язання складних завдань.

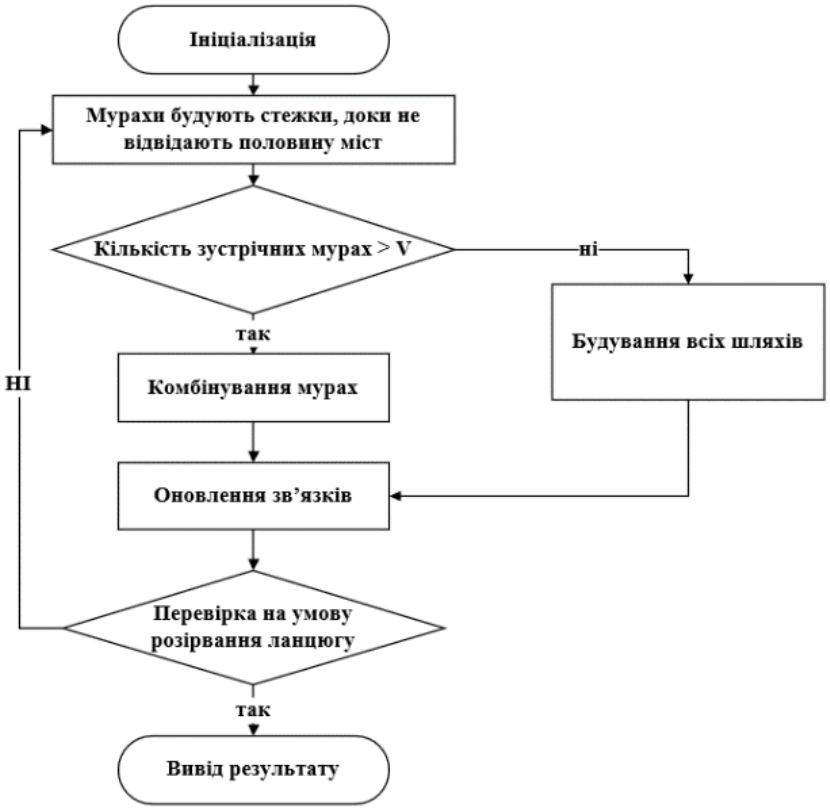


Рис. 1.14. Алгоритм оптимізації колонії мурах

Однією з ключових особливостей ACO є використання феромонів, які мурахи залишають на шляхах під час переміщення. Ці феромони інформують інших мурах про вже пройдені шляхи та їх вартість. Мурахи мають тенденцію обирати шляхи з більшою концентрацією феромонів, і вони таким чином знаходять коротший шлях до джерела їжі.

Однією з переваг ACO є її здатність адаптуватися до змін у реальному часі, що робить цей метод ефективним у вирішенні задач у змінних умовах середовища. Також він може використовуватися для вирішення широкого спектру задач у різних галузях, включаючи технічні системи, маршрутизацію мереж, а також в багатьох виданнях відеоігор та робототехніці.

2 АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ВИПАДКОВОЇ ГЕНЕРАЦІЇ РІВНІВ ДЛЯ ГРИ В ЖАНРІ TOWER DEFENSE

2.1 Аналіз та постановка проблеми існуючих методів для вирішення поставленої задачі

Отже для створення власного методу генерації рівнів для гри в жанрі Tower Defense було оглянуто варіації реалізації автоматизації та генерації окремих частин алгоритму створення рівнів. Оглянувши методи генерації карти можна зробити таку таблицю недоліків та переваг оглянутих методів.

Таблиця 2.1

Порівняльна характеристика методів генерації мапи для рівня

Метод	**Переваги**	**Недоліки**
Метод відхилення середини	Рекурсивна природа алгоритму дозволяє легко створювати деталізовані елементи ландшафту	Можливі помилки в генерації Низька точність
Метод генерації за допомогою клітинного автомату	Проста реалізація Низька складність коду та вимоги до ресурсів	Деякі конфігурації клітинних автоматів можуть призводити до нерівномірності або недосконалості генерованого ландшафту Не збалансовані
Алгоритм «Квадрат-ромб»	Швидкість	Складність реалізації

Продовження таблиці 2.1

Порівняльна характеристика методів генерації мапи для рівня

Метод	**Переваги**	**Недоліки**
Метод шуму Перліна	Гладкість та безперервність Параметризована генерація Проста реалізація	Чутливість до параметрів Складність використання на великих масштабах
Метод шуму Сімплекса	Гладкість та безперервність Параметризована генерація Проста реалізація	Висока складність створення

Аналізуючи поставлену задачу розробки методу для генерації рівнів для ігор в жанрі Tower Defense, можна зробити висновок, що оптимальним для генерації мапи алгоритмом буде саме за допомогою методу шуму Перліна. Оскільки цей метод генерації поєднує в собі ідеальні умови для виконання поставленої задачі та його недоліки або не мають великого впливу на фактичний результат, адже розмір створюваної мапи є не великим, а параметри можна легко сформувати завдяки ігровою рушію Unity.

Інші ж методи генерування мап не відповідають поставленій задачі, або ж є більш складними в реалізації, маючи при тому достатню кількість недоліків, щоб не брати їх як перспективний варіант розробки методу.

Для аналізу реалізації автоматизованого методу пошуку шляху для супротивників в грі жару Tower Defense було розглянуто такі методи як Алгоритм A*, Алгоритм Дейкстри, Генетичний алгоритм та Техніка оптимізації колонії мурах. Відповідно до проведеного аналізу їхніх особливостей роботи можна сформувати наступну таблицю переваг та недоліків вище вказаних методів:

Таблиця 2.2.

Порівняльна характеристика методів пошуку шляху

Метод	**Переваги**	**Недоліки**
Алгоритм A*	• Завжди знайде рішення і найкоротший шлях; • Зосереджується лише на цільовому вузлі; • Розширює найменшу кількість вузлів, щоб знайти оптимальний шлях. • Надійність	• Складний у виконанні; • Не оптимально, коли існує декілька цілей; • Споживає багато пам'яті; • Має проблеми зі складністю, оскільки швидкість виконання сильно залежить від точності евристики.
Алгоритм Дейкстри	• можна використовувати для вирішення великих проблем • використовується в маршрутизації, роботизованому виявленні шляху, файловому сервері тощо • Низька складність оптимальний та рівномірний • Майже лінійний і працює як для спрямованого, так і для неорієнтованого графа.	• Витрачайте час на сліпий пошук • Не може впоратися з негативними краями • Приводить до ациклічних графів
Генетичний алгоритм	Висока швидкість конвергенції.	• Залежить від початкової популяції. • Легко впасти в передчасну конвергенцію;
Техніка оптимізації колонії мурах	• Проста реалізація; • Легко розпаралелюється для одночасної обробки; Без похідних.	• Розподіл ймовірностей змінюється на ітерації; • Час для конвергенції невизначений.

Провівши аналіз найбільш використовуваних методів для пошуку шляху було виявлено, що для поставленої задачі найбільше підходить Алгоритм A* за рахунок своєї оптимальності відносно поставленого завдання: розробити метод для створення рівня для гри Tower Defense. При тому завданням для розробки буде оптимізувати A* алгоритм для того, щоб його слабкі сторони не мали суттєвого впливу на розробляємий метод.

До того ж було виявлено, що використання алгоритмів пошуку шляху на прикладі техніки оптимізації колонії мурах та алгоритму Дейкстри є не достатньо доцільним через іншу направленість задачі.

Генетичний алгоритм також є не поганим вирішенням проблеми але через свою високу ресурсо-затратність є гіршим варіантом ніж оптимізований A* алгоритм.

2.2 Виявлення можливих шляхів подальшого розвитку генерації рівнів для ігор жанру Tower Defense

Проведений аналіз методів генерації рівнів для ігор жанру Tower Defense дозволив ідентифікувати кілька напрямків подальшого розвитку цієї області.

Наразі, існуючі методи виявились не досконалими через певні недоліки, такі як повільна швидкість генерації при обробці великих об'ємів або можливість повторення вже існуючих рівнів. Шляхи оптимізації алгоритмів випадкової генерації та їхнє удосконалення для уникнення цих проблем мають великий потенціал у подальшому розвитку.

Також, подальшим напрямком у розвитку може стати використання гібридних методів генерації, які поєднують у собі кращі аспекти різних підходів. Наприклад, поєднання методів шуму Перліна та алгоритму A* може дозволити створити більш ефективні та унікальні рівні з точки зору генерації ландшафту та пошуку оптимального шляху для супротивників.

Також важливим аспектом покращення вже існуючого методу буде розвиток адаптивних систем генерації рівнів, які будуть здатні враховувати дії гравців та їхню взаємодію з грою. Це може включати реакцію на стилі гри гравця,

персоналізацію рівнів під конкретного гравця чи зміну ландшафту в залежності від розвитку гри.

Враховуючи швидкий розвиток технологій, використання штучного інтелекту, машинного навчання та глибинного навчання може відкрити нові можливості для генерації рівнів у жанрі Tower Defense. Впровадження цих передових технологій може поліпшити точність, швидкість та різноманітність генерованих рівнів.

Для поставленої задачі і проаналізованих даних оптимальним шляхом покращення створюваного методу є створення генерації мапи за рахунок зміненого під виділену мету методу шуму Перліна та покращеного A* алгоритму.

Позбутися таких недоліків методу шумів Перліна як:

- Чутливість до параметрів;
- Складність використання на великих об’ємах.

Проблема чутливості до параметрів вирішується шляхом використання ігрового рушія Unity який дозволяє оптимізувати код так, щоб більш точно корегувати вводимі данні.

Проблему зі створенням велико-об’ємних мап можна просто не вводити в розрахунки створюваного методу, так як габарити створюваних рівнів для ігор Tower Defense не під полягають даному поняттю, а отже дана проблема в поставленій задачі не є суттєвою.

Також є перспективним використання алгоритму A* для пошуку шляху на вибудуваній мапі. Серед проблемних питань даного методу було виявлено такі пункти як:

- Складний у виконанні;
- Не оптимально, коли існує декілька цілей;
- Споживає багато пам'яті;
- Має проблеми зі складністю, оскільки швидкість виконання сильно залежить від точності евристики.

Вирішенням наявних проблем є оптимізація алгоритму до поставленої задачі шляхом спрощення пошуку шляхів і обмеженням варіації евристичних елементів

до 3 направлень. Тим самим зменшуючи навантаження та пришвидшуючи процес обробки пошуку оптимального шляху.

Також не є виключенням використання в розробці інших методів, які могли бути не проаналізованими в процесі збору інформації щодо встановленої задачі і їх поєднання з вже запропонованими покращеннями створення генерації рівнів.

2.3 Формування математичної моделі створення випадкових рівнів для гри жанру Tower Defense

Оглянувши наявні алгоритми та методи для оптимізації створення рівнів в іграх жанру Tower Defense можемо перейти до формування математичної моделі розробляємого методу.

Отже для генерації самої мапи є доцільним використання методу шуму Перліна, який базується на створенні виводу рівнів висоти за рахунок функції. Основна ідея полягає в тому, щоб створити серію випадкових чисел з початковими параметрами та плавно інтерполювати між термінами в серії, заповнюючи прогалини.

Приблизний процес створення рівнів та мапи пояснює наступне зображення:

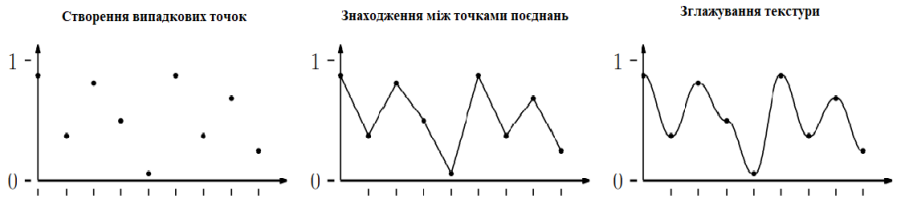


Рис. 2.1 Поетапна робота шуму Перліна

Відповідно до цього $P=(x,y)$,i=floor(x),j=floor(y) визначає точку P на мапі, де x та y - це координати точки, i та j - цілі числа, що представляють вершини тайла(квадратна або прямокутна графічна чи геометрична одиниця, яка використовується для створення поверхні чи мапи в ігровому середовищі) на мапі.

Потім використовуються градієнти $g00$,$g10$,$g01$,$g11$ та інтерполяцію для отримання значень шуму Перліна $n00$,$n10$,$n01$,$n11$ для кожної вершини тайла.

Після цього, з використанням лінійної інтерполяції, обчислюються значення

шуму для кожної точки *P* на мапі.

Використовуючи функцію `noise(nx, ny)`, вона отримує значення шуму для кожної координати `nx` та `ny`, які масштабуються від -0.5 до 0.5 за допомогою формул `nx = x/width - 0.5` та `ny = y/height - 0.5`, де `x` та `y` - це координати точки у масиві, а `width` та `height` - розміри масиву.

Формула генерації шуму Перліна у цьому випадку виглядає наступним чином, що спочатку, `nx` та `ny` масштабуються :

$$nx = \frac{x}{width},\ ny = \frac{y}{height}, \tag{2.1}$$

Потім викликається функція `noise(nx, ny)`, яка генерує значення шуму Перліна для заданих координат `nx` та `ny`. У функції `noise`, значення шуму перетворюється з діапазону від -1 до 1 до діапазону від 0 до 1 шляхом використання формули:

$$\overrightarrow{g_{00}} = grad(i,j), \overrightarrow{g_{10}} = grad\,(i+1,j) \tag{2.2}$$

$$\overrightarrow{g_{01}} = grad(i,j+1), \overrightarrow{g_{11}} = grad(i+1,j+1), \tag{2.3}$$

$$u = x - i, v = y - j, \tag{2.4}$$

$$n_{00} = \overrightarrow{g_{00}} \cdot \begin{bmatrix} u \\ v \end{bmatrix},\ n_{10} = \overrightarrow{g_{10}} \cdot \begin{bmatrix} u-1 \\ v \end{bmatrix}, \tag{2.5}$$

$$n_{01} = \overrightarrow{g_{01}} \cdot \begin{bmatrix} u \\ v-1 \end{bmatrix},\ n_{11} = \overrightarrow{g_{11}} \cdot \begin{bmatrix} u-1 \\ v-1 \end{bmatrix}, \tag{2.6}$$

$$n_{x0} = n_{00}\big(1 - f(u)\big) + n_{10} f(u), n_{x1} = n_{01}\big(1 - f(u)\big) + n_{11} f(u), \tag{2.7}$$

$$n_{xy} = n_{x0}\big(1 - f(v)\big) + n_{x1} f(v), \tag{2.8}$$

де:

$g00, g10, g01, g11$ - це градієнти шуму для чотирьох сусідніх вершин;

$v=y-j$ представляють різниці між координатами x та y і цілими частинами i та j відповідно;

$n00, n10, n01, n11$ - скалярні добутки між градієнтами та відповідними відстанями між (x,y) та кожною з вершин сітки;

$nx0, nx1, nxy$ - результати лінійної інтерполяції між $n00, n10$ та $n01, n11$ для x-координати та y-координати відповідно. Після чого отримане значення шуму записується в масив `значення` за відповідними індексами `y` та `x`.

Отже, формула для отримання значень шуму Перліна для кожної точки в

двовимірному масиві виходить з використання функції `noise(nx, ny)` з масштабуванням координат `nx` та `ny` від -0.5 до 0.5 відповідно до розмірів масиву.

Згенерувавши ландшафт мапи нам необхідно отримати шлях через який будуть просуватися атакуючі персонажі. Для цього було обрано найбільш підходящий для цього завдання алгоритм A*. Відповідно після генерації самої мапи відбувається прокладання маршруту: використовуючи алгоритм A*, для кожної клітинки на мапі визначається шлях від початкової точки до кінцевої, де $f(n)$ - сума ваги шляху $g(n)$ та евристичної вартості $h(n)$.

Отже за запропонованим методом покращення створення випадкового рівня для гри в жанрі Tower Defense відбувається за рахунок створення мапи на основі масиву даних за рахунок методу Перліна та прокладанні на цьому нового випадкового маршруту за оптимізованим A* алгоритмом, що повинен зберігати баланс та приходити до кінцевої випадкової точки.

2.4 Критерії правильної генерації рівня

Окремою задачею є формулювання правильної генерації рівня. Для цього сформуємо математичну модель обчислення для сформованої карти.

Нехай, рівень вважається правильно згенерованим, якщо всі встановлені для нього критерії було виконано. Тоді сформулюємо критерії правильної генерації мапи.

2.4.1 Координати початку та кінця шляху

Згенерований шлях має початкову точку A та точку виходу B. Якщо критерій є виконаним змінна W набуває значення 1, якщо шлях не вдалось згенерувати з обома точками виходу змінна набуває значення 0.

$$W = \begin{cases} 1, \text{якщо в шляху є точка входу та виходу,} \\ 0, \text{якщо бодай одна з них відсутня.} \end{cases}, \qquad (2.9)$$

де:

W – Критерій початкових координат;

2.4.2 Відсутність розривів

На згенерованому шляху не має бути поля для встановлення башт і він повинен бути безперервним. Якщо шлях має розриви, змінна G зменшується на 1 кожен раз, коли в шляху існує розрив. Початкове значення змінної = 0.

$$G = G_{start} + n, \tag{2.10}$$

де:

G – сума розривів;

G_{start} – початкове значення кількості розривів, яке рівне 0;

n – кількість розривів шляху.

Отже якщо значення критерію розривів досягне від’ємного значення

$$\mathrm{C} = \begin{cases} 0, \text{якщо } G > 0 \\ 1, \ G = 0; \end{cases}, \tag{2.11}$$

де:

C – значення критерію розривів.

2.4.3 Наявність рельєфу

На згенерованій мапі повинен бути рельєф. Рельєфом можна вважати якщо бодай 2, або більше, об’єкти мапи знаходяться за різнимими координатами висоти. Відповідно до цього, критерій перевірки наявності рельєфу набуває значення 1, якщо він є. Та 0 – за його відсутності.

$$R = \begin{cases} 1, \text{якщо рельєф є}, \\ 0, \text{якщо рельєф не сформований.’} \end{cases} \tag{2.12}$$

де:

R – Результат критерію наявності рельєфу.

2.4.4 Довжина шляху

Довжина шляху. Для збереження балансу гри і рівномірного розміщення башт гравця поле для цього не повинно займати менше місця аніж шлях по якому йдуть вороги. Тому його довжина не повинна перебільшувати половину ігрових клітинок мапи. Отже критерій буде набувати значення 0, якщо рівень генерується з шляхом більшим аніж половина клітинок, або меншим ніж висота мапи. Та відповідно 1 якщо його згенеровано у відповідних обмеженнях розміру.

$$L = \begin{cases} 0, \text{якщо } l > \frac{n}{2} \text{ або } l < h \\ 1, \text{інакше} \end{cases}, \quad (2.13)$$

де:

L - критерій довжини шляху;

l – довжина шляху;

n – кількість клітинок мапи;

h - довжина мапи.

Відповідно до сформованих критеріїв, якщо всі вони будуть набувати значення 1 – то рівень згенеровано правильно. Якщо хоча б 1 критерій набуває значення одиниці – значить рівень згенерувався неправильно та є помилковим.

Отже в ході розробки математичної моделі для процесу генерації рівнів для ігрового середовища Tower Defense визначено три основні критерії, які спрямовані на забезпечення якості та оптимізації створеного рівня.

Перший критерій, пов'язаний з відсутністю розривів в шляху, використовує змінну G для відстеження та контролю кількості розривів. Зазвичай, вартість G збільшується при наявності розривів, і при досягненні від'ємного значення, цей критерій приймає значення 0, що вказує на неправильно згенерований рівень.

Другий критерій, пов'язаний із наявністю рельєфу на мапі, використовує змінну R, яка набуває значення 1 при наявності рельєфу та 0 у протилежному

випадку.

А третій критерій стосується довжини шляху, який визначає, чи відповідає шлях певним обмеженням розміру. В залежності від обсягу мапи та вимог, цей критерій розглядає довжину шляху та порівнює її з певними межами, уникаючи недооцінки або перебільшення розміру шляху.

Всі ці критерії разом визначають правильність та якість згенерованого рівня. Якщо хоча б один критерій набуває значення 0, це вказує на потенційну помилку в генерації рівня, в той час як усі три критерії, приймаючи значення 1, підтверджують правильність створеного рівня.

Отже, отримана математична модель створення випадкового рівня для гри в жанрі Tower Defense базується на методі Перліна для генерації ландшафту мапи та використанні алгоритму на основі A* для прокладання оптимального маршруту.

Використання шуму Перліна для генерації дозволяє створити рельєфну мапу, яка відображає різноманітність у межах гри.

Завдяки оптимізованому A* алгоритму прокладання маршруту відбувається визначення шляху через мапу, що забезпечує правильне розміщення шляху та пересування ворожих атакуючих персонажів у грі.

Враховуючи критерії відсутності розривів, наявності рельєфу та довжини шляху, модель гарантує створення випадкового рівня, що відповідає вимогам балансу гри та забезпечує належну взаємодію між гравцем та ворожими силами.

Таким чином, запропонована математична модель випадкового генерування рівнів в грі Tower Defense є ефективним рішенням для створення різноманітних та цікавих ігрових сценаріїв.

Впровадження розробленого методу повинно не лише генерувати карту з рельєфом, але й визначати оптимальний шлях для руху ворожих сил, що значно збагачує ігрову механіку. Отримана модель повинна реалізовувати критерії безперервності шляху, наявності рельєфу та відповідності розміру шляху в межах заданих обмежень, забезпечуючи стабільний і ефективний процес генерації рівнів. Такий підхід дає можливість в грі Tower Defense отримати різноманітні, випадкові та водночас збалансовані рівні.

3 РОЗРОБКА МОДЕЛІ ВИПАДКОВОЇ ГЕНЕРАЦІЇ РІВНІВ ДЛЯ ГРИ В ЖАНРІ TOWER DEFENSE

3.1 Опис використаних програмних засобів

Створення рівнів для гри в жанрі Tower Defense здебільшого навіть в оглянутих джерелах відбувається за рахунок ігрового рушія Unity. Такий вибір обумовлюється наявністю всіх необхідних інструментів для розробки ігор, доступністю рушія як програмно, так і в фінансовому плані, бо він є безкоштовним.

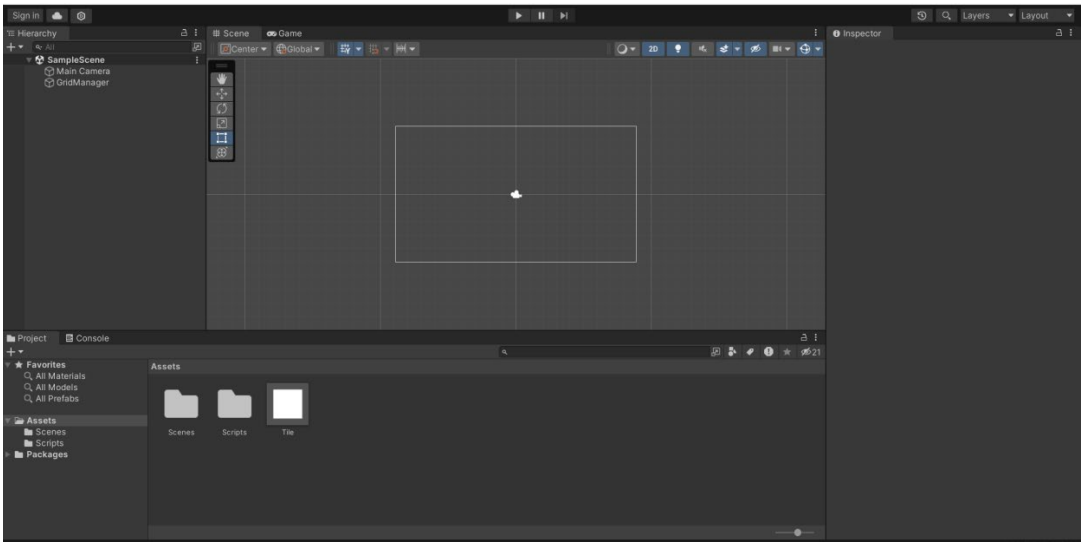


Рис. 3.1. Ігровий рушій Unity

Unity - це крос-платформенний гральний рушій та інтегроване середовище розробки, створене для створення інтерактивних 2D та 3D додатків, головним чином у галузі відеоігор, але також використовується у симуляторах, тренувальних програмах, архітектурній візуалізації та інших областях. Unity дозволяє розробникам створювати додатки, які можуть працювати на різних платформах, таких як комп'ютери, мобільні пристрої, консолі, віртуальна реальність та навіть веб-браузери.

Unity має потужний графічний рушій, підтримку фізики, звуку, анімацій та

багатьох інших функцій, що спрощують процес створення ігор та інших візуальних додатків. Також надає широкий набір інструментів для управління ресурсами, створення сцен, програмування грального логіки та взаємодії з користувачем. Також до переваг Unity можна віднести:

Кросплатформенність;

Модульність та розширюваність;

Широкий функціонал;

Можливість оптимізації.

Завдяки цим перевагам вибір на користь саме цього рушія є очевидною. Відповідно обрана мова програмування C# а середа для написання коду – Visual Studio.

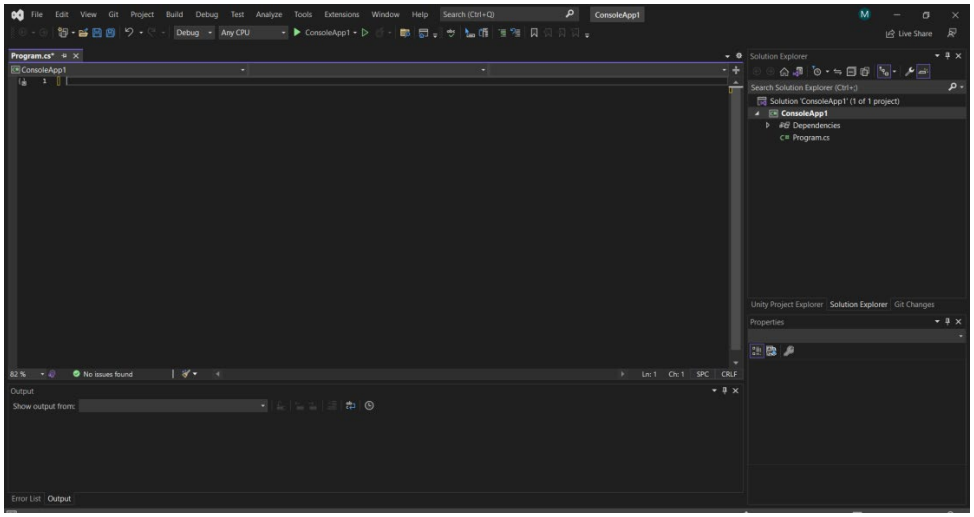


Рис. 3.2. Microsoft Visual Studio

Об'єктно-орієнтована мова C#, яку розробив Microsoft. Вона використовується для розробки різноманітних програм, включаючи ігри, додатки для Windows, веб-сайти та багато іншого. Це мова загального призначення, розроблена для розробки додатків на платформі Microsoft, для роботи якої потрібна платформа .NET у Windows. C# часто вважають гібридом, який використовує найкраще з C і C++ для створення справді модернізованої мови. Хоча платформа .NET підтримує кілька інших мов кодування, C# швидко стала однією з найпопулярніших.

C# широко використовується для створення ігор за допомогою ігрового движка Unity , який сьогодні є найпопулярнішим ігровим движком. Більше третини найкращих ігор створено за допомогою Unity, і є приблизно 770 мільйонів активних користувачів ігор, створених за допомогою механізму Unity.

3.2 Опис структури проекту

Відповідно на початку алгоритм формує точки зв’язку. Після чого поєднує їх прямим сполученням і завершує своє виконання згладжуванням створюваної поверхності мапи. Згладжування відбувається за рахунок інтерполяції – тобто знаходженням проміжних відрізків для скорочення відстані між існуючими точками.

Рис. 3.3. Алгоритм генерації мапи за методом шумів Перліна

Для кожної клітини вираховується випадкова точка мапи та обчислюються

випадкові вектори від крайніх точок створеної клітини. Ці вектори визначаються як випадкові напрямки, що виходять з крайніх точок, надаючи кожній клітині свій характерний шумовий вигляд.

Після цього, розраховується різниця векторів від кута до обраної точки, використовуючи заздалегідь згенеровані випадкові значення. Це дозволяє визначити напрямок шуму для кожної точки мапи, що забезпечує різноманітність мапи.

Третій етап включає скалярний добуток цих отриманих векторів, що дозволяє отримати значення шуму для кожної точки. Це значення шуму інтерполюється в межах кожної клітини, що дозволяє створити плавну та згладжену мапу зміни шуму.

Такий процес дозволяє створювати карту з непередбачуваними топографічними особливостями, що відтворюють натуральний вигляд, а також забезпечують різноманітність і випадковість для створення захоплюючого ігрового середовища для створюваної гри в жанрі Tower Defense.

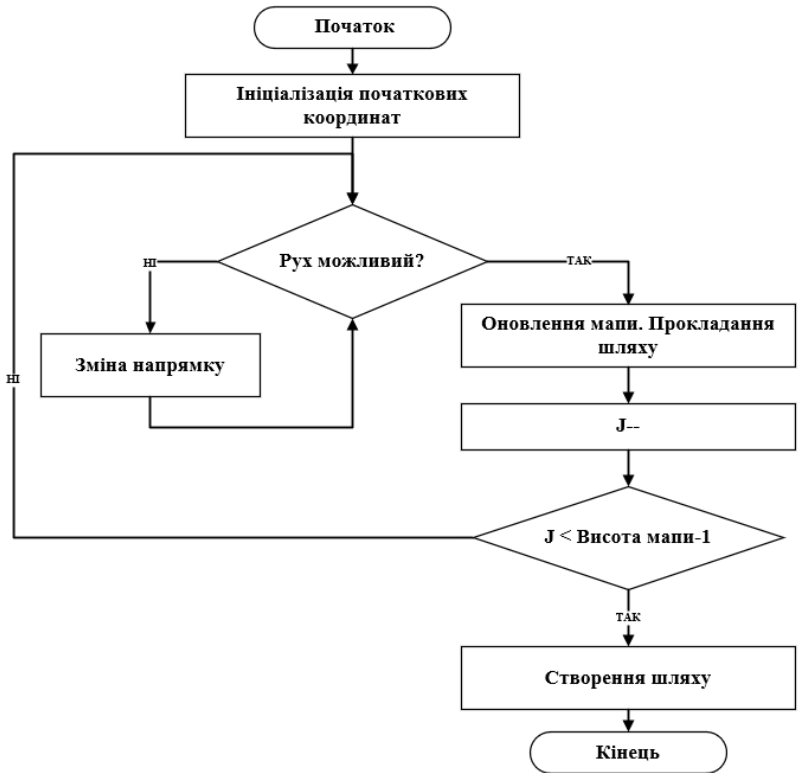


Рис. 3.4. Алгоритм генерації шляху на основі алгоритму A*

Початок руху означає, що об'єкт, який рухається по мапі, стартує з певної

початкової позиції і спрямовується у визначений напрямок. Якщо рух у даному напрямку неможливий (наприклад, зустрічається перешкода), алгоритм виконує спробу змінити напрямок руху - повернути на ліво або право.

Якщо об'єкт, що рухається, змушений повертати в одному напрямку більше трьох разів підряд (наприклад, три рази вліво), алгоритм вносить зміни та здійснює поворот в іншому напрямку (наприклад, вправо), щоб уникнути закріпленості в одному напрямку.

Нехай T - максимальна кількість дозволених послідовних поворотів в одному напрямку (наприклад, ліворуч). Нехай N - фактична кількість поворотів в одному напрямку на даний момент. Якщо N > T, то змінити напрямок руху на протилежний.

Це може бути виражено у вигляді умови:

Якщо N > T, то змінити напрямок руху.

Де N - лічильник поворотів в одному напрямку, який збільшується при кожному новому повороті, і T - максимальна допустима кількість послідовних поворотів в одному напрямку.

Цей процес руху по мапі продовжується доти, доки об'єкт, який рухається, не дійде до кінця мапи або поки не будуть виконані інші критерії зупинки. Наприклад, у залежності від умови, що визначена для завершення руху, алгоритм може припинити переміщення об'єкта після досягнення певної точки, визначеної кінцем мапи або іншими обмеженнями.

Нехай P - позиція об'єкта на мапі, а E - кінцева точка (кінець мапи або будь-яке інше обмеження). Повинна виконуватись умова:

Якщо $P=E$, то зупинити рух об’єкта

Це означає, що якщо поточна позиція об'єкта (P) дорівнює кінцевій точці (E), то рух об'єкта припиняється, оскільки ціль досягнута.

Тут P і E можуть бути векторами позицій на мапі, а порівняння P = E вказує на те, що об'єкт досягнув кінцевої позиції, яка була задана як критерій завершення руху.

Таким чином алгоритм A* буде відповідати поставленим умовам та зберігати баланс генерації рівня.

Відповідно до запропонованих та розроблених нових методів генерації мапи та автоматичного прокладання маршруту алгоритм створення рівня тепер набуває такого вигляду:

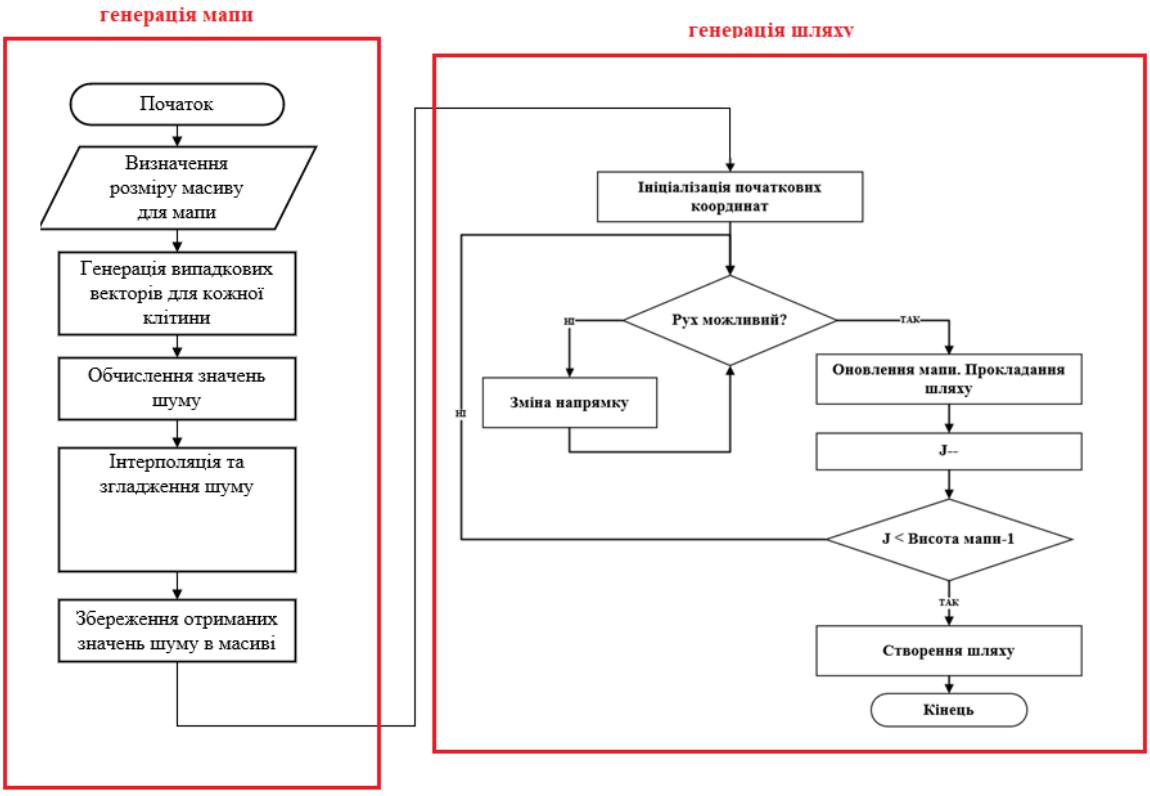


Рис. 3.5 Алгоритм генерації випадкового рівня

Сформульований алгоритм створення випадкового рівня в жанрі Tower Defense використовує набір процедур для вирішення важливих завдань у грі. Передусім, він починається з ініціалізації масиву даних, де зберігаються основні характеристики мапи, такі як рельєф, місцеположення важливих об'єктів та територій, де буде розгортатися гра.

Далі йде процес генерації мапи, що полягає в створенні структури мапи за допомогою визначених алгоритмів. Цей процес враховує багато параметрів, включаючи структуру території, розташування ключових елементів, таких як ворожі точки або захисні об'єкти, і формування природних областей, які роблять мапу цікавою та викликають захоплення у гравців.

Останнім кроком є автоматичне прокладання маршруту, де застосовуються алгоритми, які визначають шляхи руху для ворожих одиниць або об'єктів у грі. Цей процес створює випадковий, проте оптимальний маршрут, забезпечуючи баланс між геймплеєм та викликаючи цікаві стратегічні рішення у гравців. Зокрема, такий маршрут веде до кінцевої точки на мапі, яка також генерується випадковим чином, сприяючи унікальності та варіативності кожної гри.

Отже за запропонованим методом покращення створення випадкового рівня для гри в жанрі Tower Defense відбувається за рахунок створення мапи на основі масиву даних за рахунок методу Перліна та прокладанні на цьому нового випадкового маршруту за оптимізованим A* алгоритмом, що повинен зберігати баланс та приходити до кінцевої випадкової точки.

3.3 Опис інтерфейсу

Для розробляємого додатку, що буде реалізовувати створений метод було сформовано наступний вигляд інтерфейсу:

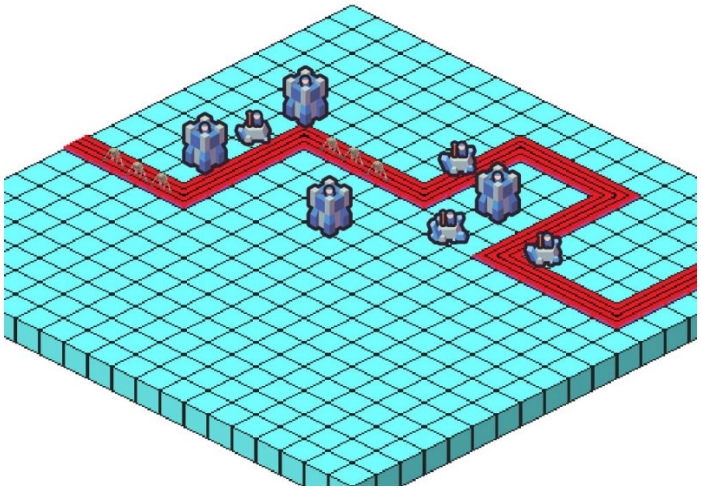


Рис. 3.6. Інтерфейс рівня

Де загальний розмір карти налаштовується напряму в Uniry тим самим умикаючи недоліка в методі шумів Перліна з необхідністю в чітких початкових даних.

Для цього було використано вікно "Інспектор" (Inspector) у Unity, що дозволяє переглядати та редагувати всі властивості обраного на даний момент об'єкта (GameObject). Оскільки різні типи GameObject мають різні набори властивостей, макет та вміст вікна Інспектора змінюються кожен раз при виборі іншого GameObject.

У цьому вікні можна бачити та налаштовувати параметри, які визначають обрану об'єктом сутність. Наприклад, якщо обрано об'єкт світла, у вікні Інспектора можна регулювати його яскравість, колір, тип світла тощо. Якщо обрано об'єкт гравця, ви зможете бачити та змінювати його характеристики, такі як швидкість руху, здоров'я, сила тощо.

Ця функція надає зручний інтерфейс для взаємодії з різними елементами гри, дозволяючи швидко переглядати та змінювати їх властивості безпосередньо в редакторі Unity.

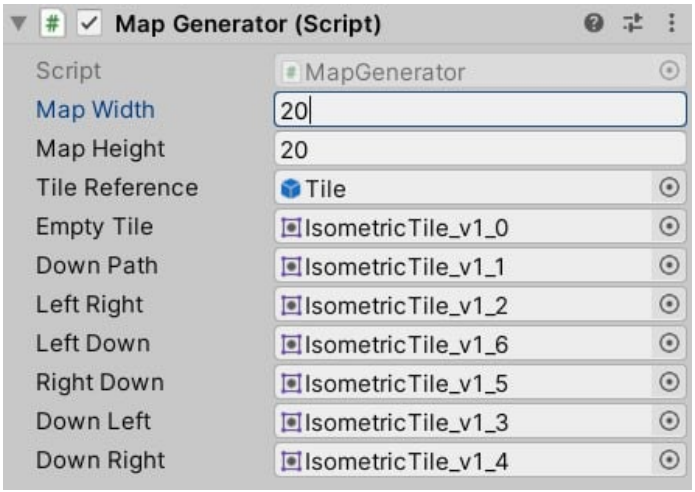


Рис. 3.7. Використання вікна Інспектор для налаштувань рівня

Як можна бачити на рисунку, задані розміри карти – 20 на 20 клітинок. Також для створення рівня важливу роль відіграють обрані елементи які використовувались для ілюстрації генерованої карти, башти та вороги.

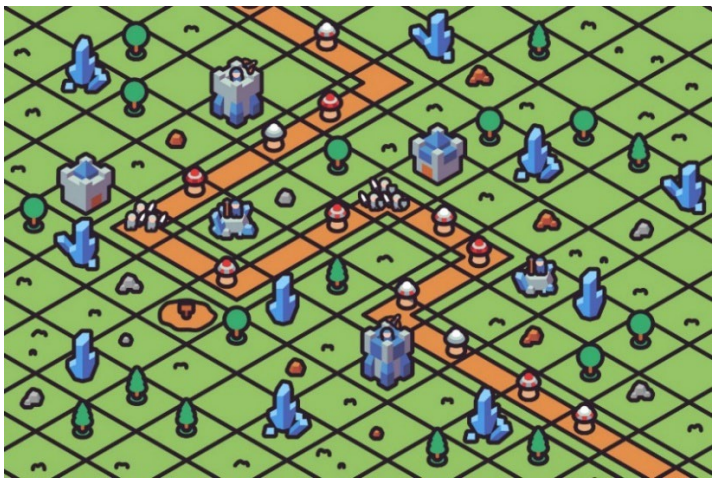


Рис. 3.8. Спрайти башт та мапи рівня

Використані спрайти башт мають чотири види веж. А саме:

- Стрілецька;
- Казарма;
- Катапульта;
- Чарівник.

А також кожна вежа має три рівні покращення, що є важливим для створюваного зовнішнього вигляду розробляємого рівня в грі жанру Tower Defense.

Для генерації ворогів, що будуть пересуватись мапою було використано інший набір спрайтів і вони мають наступний вигляд:

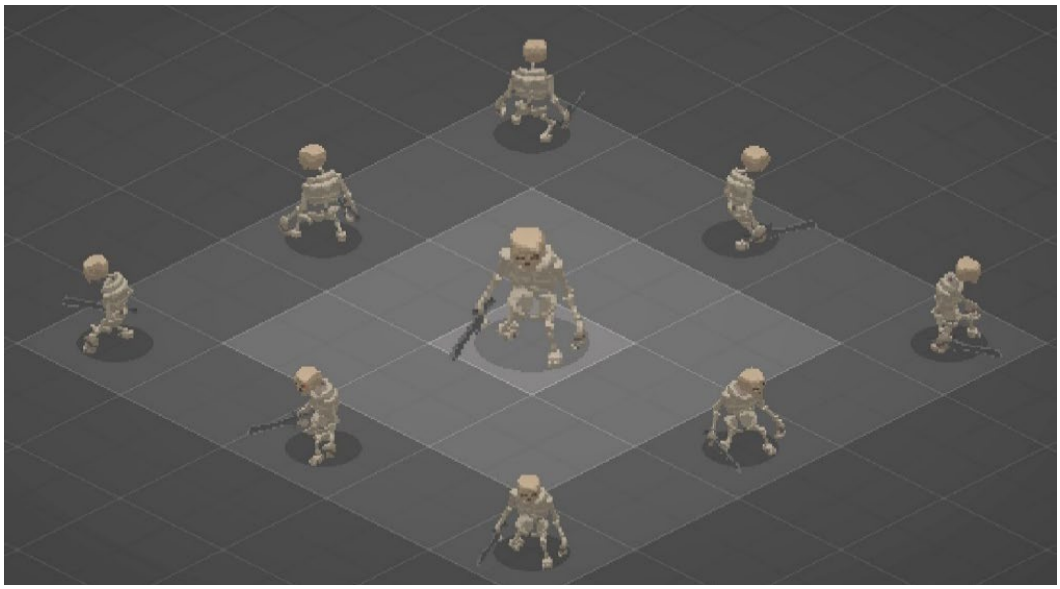


Рис. 3.9. Вигляд супротивників

3.4 Опис розроблених класів

Код MapGenerator визначає клас у середовищі Unity для генерації та управління картою в грі. Задаються розміри карти, префаб плитки та різні спрайти для шляхів, які можна налаштовувати в інспекторі Unity. Змінні curX та curY відслідковують поточні координати генератора. Змінні forceDirectionChange, continueLeft та continueRight використовуються для управління напрямком шляху.

Для створення генератора карт в ігровому середовищі Unity , було сформовано наступний код:

```
using UnityEngine;

public class MapGenerator : MonoBehaviour
{
    // Розміри карти
    [SerializeField] private int mapWidth, mapHeight;

    // Посилання на префаб плитки та різні спрайти для шляху
    [SerializeField] private GameObject tileReference;
    [SerializeField] private Sprite emptyTile, downPath, leftRight, leftDown, rightDown, downLeft, downRight;

    // Поточні координати та спрайт для використання
    private int curX;
    private int curY;
    private Sprite spriteToUse;
    private bool forceDirectionChange = false;

    // Змінні для слідування шляху вліво та вправо
    private bool continueLeft = false;
    private bool continueRight = false;
    private int currentCount = 0;

    // Перечислення для визначення напрямку
    private enum CurrentDirection
    {
        LEFT,
        RIGHT,
        DOWN,
        UP
    };
    private CurrentDirection curDirection = CurrentDirection.DOWN;
```

Рис. 3.10. MapGenerator

Основна ідея полягає в динамічному створенні картини, де шлях може змінювати напрямок, обходячи перешкоди та визначаючи свій власний маршрут.

Створюється масив tileData, який визначає структуру даних для кожної плитки на карті. Кожна плитка має свої координати, ідентифікатор та спрайт. Використовуючи цей масив, генерується початкова частина карти, де кожна плитка представлена у вигляді графічного об'єкта (префабу) з використанням відомих спрайтів.

Метод GeneratePath використовується для поетапної генерації шляху вниз по

карті. Під час генерації вирішується, в якому напрямку буде рухатися шлях. Передбачено логіку для зміни напрямку шляху внаслідок різних умов, таких як перешкоди або генерація випадкового напрямку. Також сформована логіка, яка коригує координати в разі зміни напрямку, щоб уникнути неправильного розташування плиток.

Було вирішено для покращення роботи програми використовувати корутини та затримки для поетапної генерації шляху з визначеною затримкою для кращої візуалізації.

Основна логіка зосереджена в методі GeneratePath, який, використовуючи корутини та логіку зміни напрямку, генерує шлях по карті. Методи UpdateMap, CheckCurrentDirections, ChooseDirection, GoUp, GoLeft та GoRight служать для зміни стану картини та управління рухом.

Присутні коректори використання масиву tileData, який ініціалізується у методі Awake. Кожен крок генерації шляху включає оновлення координат, визначення напрямку, зміну флагів та відображення шляху на карті з використанням різних спрайтів.

У наступному фрагменті коду визначено структуру TileData, яка використовується для зберігання даних про кожну плитку на карті. Також, є ініціалізація та генерація карти у методі Awake, який викликається при старті гри.

```
// Структура для збереження даних про плитку
public struct TileData
{
    public Transform transform;
    public SpriteRenderer spriteRenderer;
    public int tileID;
}

// Масив даних про плитки
TileData[,] tileData;

void Awake()
{
    // Ініціалізація масиву даних та генерація карти
    tileData = new TileData[mapWidth, mapHeight];
    GenerateMap();
}

void GenerateMap()
{
    // Генерація основної частини карти
    for (int x = mapWidth - 1; x >= 0; x--)
    {
        for (int y = 0; y < mapHeight; )
        {
            float xOffset = (x + y) / 2f;
            float yOffset = (x - y) / 4f;
            GameObject newTile = Instantiate(tileReference, new Vector2(xOffset, yOffset), Quaternion.identity);
            tileData[x, y].spriteRenderer = newTile.GetComponent<SpriteRenderer>();
            tileData[x, y].tileID = 0;
            tileData[x, y].spriteRenderer.sprite = emptyTile;
            tileData[x, y].transform = newTile.transform;
        }
    }
    StartCoroutine(GeneratePath());
}
```

Рис. 3.11. TileData

Цей код визначає структуру для збереження даних про плитку на карті. Також він містить посилання на об'єкт Transform, SpriteRenderer та ідентифікатор плитки (tileID).

Ініціалізація масиву tileData відбувається в методі Awake створюється двовимірний масив tileData з розмірами карти mapWidth та mapHeight. Після ініціалізації масиву викликається метод GenerateMap для генерації карти.

Використовуючи вкладені цикли for, генерується основна частина карти від нижнього правого кута до верхнього лівого. Кожній плитці присвоюються координати та спрайт відповідно до формул визначення xOffset та yOffset. Створюється новий об'єкт newTile на основі префабу tileReference, розміщується в обчислених координатах. Так відбувається генерація карти в методі GenerateMap

Також на цьому фрагменті коду відбувається ініціалізація даних tileData[x, y]: Для кожної плитки в масиві tileData встановлюються значення tileID, spriteRenderer та координати.

Для подальшої генерації шляху на карті Запускається сопрограма GeneratePath.

Загалом фрагмент коду визначає структуру даних для плиток і виконує їх ініціалізацію та генерацію на початковому етапі гри.

Отже, у цій частині коду реалізовано генератор карт для ігрового середовища Unity. Він використовує структуру даних TileData, яка зберігає інформацію про кожну плитку на карті, включаючи посилання на об'єкт Transform, SpriteRenderer та ідентифікатор плитки.

Процес генерації карт відбувається у методі GenerateMap, який визначає розмір карти, ініціалізує масив tileData та створює графічні об'єкти плиток на основі заданого префабу tileReference.

Кожній плитці присвоюється ідентифікатор, спрайт та координати з урахуванням певних обчислень. Потім викликається метод GeneratePath, який ініціює генерацію шляху на карті.

Логіка генерації шляху включає перевірки напрямку та можливості руху, зміну напрямку через перешкоди та динамічний вибір нового напрямку.

Користуючись корутинами, генератор крок за кроком візуалізує шлях зі затримкою.

У подальшій частині коду визначено корутину GeneratePath, яка відповідає за поетапну генерацію шляху на карті. Це важливий компонент для визначення динаміки генерації мапи.

```
IEnumerator GeneratePath()
{
    // Ініціалізація початкових координат та спрайту для шляху
    curX = Random.Range(0, mapWidth);
    curY = 0;
    spriteToUse = downPath;

    while (curY <= mapHeight - 1)
    {
        // Перевірка можливості продовження в поточному напрямку та вибір нового напрямку
        CheckCurrentDirections();
        ChooseDirection();

        // Оновлення карти
        if (curY <= mapHeight - 1)
        {
            UpdateMap(curX, curY, spriteToUse);
        }

        // Інкремент координат для напрямку вниз
        if (curDirection == CurrentDirection.DOWN)
        {
            curY++;
        }

        // Затримка для візуалізації генерації
        yield return new WaitForSeconds(0.05f);
    }
}
```

Рис. 3.12. GeneratePath

Ця частина коду допомагає крок за кроком генерувати та візуалізувати шлях на карті, додавати динаміку до процесу генерації та забезпечувати плавний перехід між етапами гри. Процес генерації реалізований у класі MapGenerator, який включає структуру TileData для представлення даних про кожну плитку на карті. Карта генерується у методі GenerateMap, де використовується двовимірний масив tileData.

Змінні curX та curY ініціалізуються випадковим чином для визначення

стартових координат. А spriteToUse ініціалізується спрайтом downPath, що вказує на напрямок шляху вниз.

Структура TileData містить інформацію про кожну плитку, включаючи її ідентифікатор, посилання на графічний об'єкт та інші деталі. Це дозволяє динамічно оновлювати та змінювати характеристики плиток під час гри.

Далі описується цикл генерації шляху: while-цикл триває, доки curY не досягне верхньої межі карти (mapHeight - 1). Усередині циклу викликається метод CheckCurrentDirections() для перевірки можливості продовження в поточному напрямку та метод ChooseDirection() для вибору нового напрямку.

Викликається метод UpdateMap, який відповідає за оновлення картини з урахуванням поточних координат, спрайту та напрямку.

Важливими елементами є перевірки можливості продовження шляху в поточному напрямку, вибір нового напрямку та візуалізація генерації з затримкою. Застосування корутин дозволяє створювати плавний та анімований процес генерації карт. Якщо напрямок шляху - вниз (curDirection == CurrentDirection.DOWN), curY інкрементується для переходу до наступного рядка на карті.

До того ж використовується yield return new WaitForSeconds(0.05f), щоб надати затримку між кроками генерації. Це поліпшує візуальний ефект та взаємодію гравця з генератором карт. Отже розроблений код використовує концепції об'єктно-орієнтованого програмування, включає в себе використання корутин для асинхронної генерації та затримки для візуалізації. Він створює гнучку та варіативну систему генерації ігрових карт з динамічно змінюваними шляхами.

В наступній частині коду визначено приватний метод CheckCurrentDirections, який використовується для перевірки можливості руху в поточному напрямку та відповідного оброблення цієї інформації.

```
private void CheckCurrentDirections()
{
    // Перевірка можливості руху в поточному напрямку та зміна координат
    if (curDirection == CurrentDirection.LEFT && curX - 1 >= 0 && tileData[curX - 1, curY].tileID == 0)
    {
        curX--;
    }
    else if (curDirection == CurrentDirection.RIGHT && curX + 1 <= mapWidth - 1 && tileData[curX + 1, curY].tileID == 0)
    {
        curX++;
    }
    else if (curDirection == CurrentDirection.UP && curY - 1 >= 0 && tileData[curX, curY - 1].tileID == 0)
    {
        // Перевірка можливості руху вгору та продовження в поточному напрямку
        if (continueLeft && tileData[curX - 1, curY - 1].tileID == 0 ||
            continueRight && tileData[curX + 1, curY - 1].tileID == 0)
        {
            curY--;
        }
        else
        {
            // Зміна напрямку внаслідок перешкоди
            forceDirectionChange = true;
            tileData[curX, curY].transform.position = new Vector2(tileData[curX, curY].transform.position.x, tileData[curX, curY].transform.position.y - 0.3f);
        }
    }
    else if (curDirection != CurrentDirection.DOWN)
    {
        // Зміна напрямку внаслідок перешкоди
        forceDirectionChange = true;
        tileData[curX, curY].transform.position = new Vector2(tileData[curX, curY].transform.position.x, tileData[curX, curY].transform.position.y - 0.3f);
    }
}
```

Рис. 3.13. Метод CheckCurrentDirections

Метод CheckCurrentDirections використовується для перевірки можливості руху в поточному напрямку, враховуючи можливість зміни напрямку внаслідок перешкод. Кожна зміна напрямку включає в себе корекцію координат та зміну спрайту.

В коді відбувається наступна логіка перевірки руху вліво: Якщо напрямок curDirection - ліво, і curX - 1 не виходить за межі карти і плитка пуста (tileData[curX - 1, curY].tileID == 0), то curX зменшується на одиницю.

Та відповідно до цього, якщо напрямок - право, і curX + 1 не виходить за межі карти і плитка пуста, то curX збільшується на одиницю.

Якщо напрямок - вгору, curY - 1 не виходить за межі карти і плитка пуста, перевіряється можливість продовження в поточному напрямку. Якщо існує можливість (continueLeft або continueRight), або плитка пуста, curY зменшується на одиницю.

Зміна напрямку вгору внаслідок перешкоди відбувається наступним чином:

Якщо не можливо рухатися вгору, встановлюється флаг forceDirectionChange в true, і позиція поточної плитки коригується відповідно до напрямку вниз.

А зміна напрямку вниз внаслідок перешкоди, якщо напрямок не вниз, встановлюється флаг forceDirectionChange в true, і позиція поточної плитки коригується відповідно до напрямку вгору.

Цей код створює гнучкий генератор карт змінюючихся шляхів, де використовуються принципи об'єктно-орієнтованого програмування, асинхронної обробки та візуалізації з використанням анімацій та спеціальних ефектів.

Цей метод грає важливу роль у визначенні можливості руху та корекції шляху в залежності від умов на карті під час генерації. Та реалізує описаний вище та розроблений алгоритм на основі A* .

Ключовим аспектом є логіка випадкового вибору та зміни напрямку руху шляху, що дозволяє створювати варіативні та цікаві маршрути.

```
private void ChangeDirection()
{
    // Логіка зміни напрямку руху

    // Генерація випадкового значення для вибору напрямку
    int dirValue = Mathf.FloorToInt(Random.value * 2.99f);

    // Перевірка умов для зміни напрямку на вертикальний (вгору)
    if ((dirValue == 0 && curDirection == CurrentDirection.LEFT && curX - 1 > 0)
      || (dirValue == 0 && curDirection == CurrentDirection.RIGHT && curX + 1 < mapWidth - 1))
    {
        // Перевірка додаткових умов для зміни напрямку вгору
        if (curY - 1 >= 0 &&
            tileData[curX, curY - 1].tileID == 0 &&
            tileData[curX - 1, curY - 1].tileID == 0 &&
            tileData[curX + 1, curY - 1].tileID == 0)
        {
            GoUp();  // Метод для зміни напрямку вгору
            return;
        }
    }
```

Рис. 3.14. Логіка зміни напрямку руху

Цей фрагмент коду доповнює головний алгоритм генерації шляху, дозволяючи йому динамічно змінювати напрямок руху, враховуючи обмеження та умови, що створюють гнучкий та адаптивний маршрут в ігровому середовищі.

Процес генерації шляху відбувається шляхом ітерації по карті у зворотньому

порядку за координатою X та у прямому порядку за координатою Y. При цьому кожній плитці присвоюються параметри, такі як ідентифікатор, позиція та спрайт.

Код визначає логіку обрання нового напрямку на основі випадкового вибору та умов на карті. Використовується випадкове число для вибору напрямку, яке генерується в межах від 0 до 2.99. Потім перевіряється можливість зміни напрямку на вертикальний (вгору) для поточного напрямку вліво чи вправо. У разі відповідності умовам для зміни напрямку вгору, викликається метод GoUp(), який відповідає за зміну напрямку вгору та необхідні операції. Після зміни напрямку відбувається оновлення карті та затримка для візуалізації генерації.

Логіка випадкового вибору напрямку забезпечує гнучкість генерації. Враховуючи умови, можливість зміни напрямку на вертикальний вгору активується тільки при наявності вільних клітинок у верхньому напрямку на карті

Також було введено умови для зміни напрямку вгору лише при наявності вільних клітинок у верхньому напрямку на карті.

Усі ці елементи коду спільно допомагають створювати динамічний та варіативний шлях на карті гри, забезпечуючи цікавий геймплей та взаємодію з ігровим середовищем.

Цей фрагмент коду сприяє гнучкості генерації шляху за рахунок випадкового вибору напрямку та врахування умов, які активують можливість зміни напрямку на вертикальний вгору лише у випадку вільних клітинок у верхньому напрямку на карті. Ці елементи сприяють створенню динамічного та варіативного шляху у грі, що впливає на цікавість геймплею та взаємодію з ігровим середовищем.

У цьому фрагменті коду реалізована логіка обробки руху шляху вліво та вправо на ігровій карті в контексті генерації шляху. Кожен крок у цьому відокремленому блоку коду відповідає за конкретний аспект цієї логіки.

Обробка руху вліво та вправо на ігровій карті в залежності від поточного напрямку описується в наступному фрагменті коду.

```
// Логіка для напрямку вліво
if (curDirection == CurrentDirection.LEFT)
{
    UpdateMap(curX, curY, leftDown);
}
// Логіка для напрямку вправо
else if (curDirection == CurrentDirection.RIGHT)
{
    UpdateMap(curX, curY, rightDown);
}

// Зміна напрямку вниз для лівого або правого напрямку
if (curDirection == CurrentDirection.LEFT || curDirection == CurrentDirection.RIGHT)
{
    curY++;
    spriteToUse = downPath;
    curDirection = CurrentDirection.DOWN;
    return;
}
```

Рис. 3.15. Руху шляху вліво та вправо на ігровій карті

Цей фрагмент важливий для динамічної зміни шляху та оновлення ігрового стану.

Умова перевірки напрямку: Якщо поточний напрямок руху - вліво (CurrentDirection.LEFT).

Також в цій частині відбувається виклик методу UpdateMap(): Якщо рух вліво, оновлюються дані карти за допомогою методу UpdateMap(), який визначає новий спрайт для поточної плитки.

Починаємо з перевірки поточного напрямку руху. Якщо напрямок - вліво (CurrentDirection.LEFT), викликається метод UpdateMap(), який оновлює дані карті для відповідного напрямку. Те саме стосується і випадку руху вправо (CurrentDirection.RIGHT).

Після цього перевіряється поточний напрямок руху, і якщо він вказує на ліво чи право, відбувається зміна координати Y та спрайту для відповідної плитки. Ця зміна напрямку вниз є частиною логіки генерації шляху та враховується для створення варіативності у маршрутах.

У разі зміни напрямку вниз, використовується оператор return, який припиняє

виконання методу, оскільки це важливо для контролю над подальшими етапами генерації.

Цей код відповідає за обробку руху вліво та вправо на карті та забезпечує логіку для зміни напрямку вниз, враховуючи поточні умови генерації шляху на карті гри. Отже, цей фрагмент коду взаємодіє з різними елементами генератора шляху, враховуючи поточний напрямок та забезпечуючи динамічність та цікавість створюваних шляхів на ігровій карті.

```
// Перевірка можливості зміни напрямку на горизонтальний (ліво або право)
if (curX - 1 > 0 && curX + 1 < mapWidth - 1 || continueLeft || continueRight)
{
    // Логіка для напрямку вліво
    if (dirValue == 1 && !continueRight || continueLeft)
    {
        if (tileData[curX - 1, curY].tileID == 0)
        {
            // Логіка для зміни напрямку вліво
            if (continueLeft)
            {
                spriteToUse = rightDown;
                continueLeft = false;
            }
            else
            {
                spriteToUse = downLeft;
            }
            curDirection = CurrentDirection.LEFT;
        }
    }
    // Логіка для напрямку вправо
    else
    {
        if (tileData[curX + 1, curY].tileID == 0)
        {
            // Логіка для зміни напрямку вправо
            if (continueRight)
            {
                continueRight = false;
                spriteToUse = leftDown;
            }
            else
            {
                spriteToUse = downRight;
            }
            curDirection = CurrentDirection.RIGHT;
        }
    }
}
```

Рис. 3.16. Перевірка можливості зміни напрямку

У даній частині коду реалізована логіка зміни напрямку руху шляху на горизонтальний (ліво або право) в контексті генерації ігрової карти. Давайте розглянемо цей фрагмент детальніше:

У першому блоці коду перевіряється можливість зміни напрямку на

горизонтальний шлях, а випадковим чином обирається новий напрямок - ліво або право. Починаємо з перевірки можливості зміни напрямку на горизонтальний шлях. Ця можливість враховує межі карти та попередні напрямки руху, щоб забезпечити логічний та непередбачуваний хід генерації.

Умова включає в себе перевірку, чи знаходимося ми в межах карты, чи відсутні прапорці continueLeft та continueRight, що вказують на можливість продовження руху вліво або вправо відповідно. Далі визначається випадковим чином новий напрямок за допомогою генерації випадкового числа dirValue, що дорівнює 0 або 1.

У разі, якщо випадкове число дорівнює 1 та виконується одна з двох умов:

- Якщо ми не продовжуємо вправо (!continueRight) або продовжуємо вліво (continueLeft), то перевіряється, чи можливий рух вліво. Якщо так, то змінюється спрайт та напрямок руху на вліво.

- У випадку, якщо умова не виконується, перевіряється можливість руху вправо. Якщо так, то змінюється спрайт та напрямок руху на вправо.

В кінці, визначається новий напрямок руху (curDirection) на основі змінених параметрів. Останній крок полягає у встановленні нового напрямку руху та зміні відповідного спрайту, що забезпечує різноманітність у генерації шляхів на ігровій карті.

Цей фрагмент важливий для забезпечення різноманітності та цікавості у генерації шляхів на ігровій карті, забезпечуючи зміну напрямків у відповідь на умови, які визначаються під час виконання алгоритму.

В наступній частині коду реалізована логіка зміни напрямку руху на основі умов та обраного випадкового значення. Якщо можливо рухатися вліво (з урахуванням границь карти), то встановлюється відповідний спрайт та напрямок вліво. Аналогічно, якщо можливо рухатися вправо, то встановлюється спрайт та напрямок вправо

```
    }
    // Логіка для зміни напрямку вліво, якщо можливо
    else if (curX - 1 > 0)
    {
        spriteToUse = downLeft;
        curDirection = CurrentDirection.LEFT;
    }
    // Логіка для зміни напрямку вправо, якщо можливо
    else if (curX + 1 < mapWidth - 1)
    {
        spriteToUse = downRight;
        curDirection = CurrentDirection.RIGHT;
    }


    if (curDirection == CurrentDirection.LEFT)
    {
        GoLeft();
    }
    else if (curDirection == CurrentDirection.RIGHT)
    {
        GoRight();
    }
}
```

Рис. 3.17. Продовження логіки зміни руху генерованого шляху

Після встановлення нового напрямку викликаються методи GoLeft() або GoRight(), які відповідають за реальний рух вліво чи вправо. Це може включати оновлення картографії, зміну координат, анімаційні ефекти чи будь-які інші дії, пов'язані із зміною напрямку.

Додатково, враховуються стани "continueLeft" та "continueRight", які вказують, що шлях буде продовжений вліво або вправо від поточного напрямку. Це важливо для створення зміни напрямку та надає можливість генерувати різноманітні форми шляхів.

Цей фрагмент важливий для забезпечення системи керування шляхом та його динамічності в процесі генерації гри. Такий підхід дозволяє створювати більш складні та інтерактивні шляхи на ігровій карті.

```
private void GoUp()
{
    // Рух вгору залежно від напрямку
    if (curDirection == CurrentDirection.LEFT)
    {
        UpdateMap(curX, curY, downRight);
        continueLeft = true;
    }
    else
    {
        UpdateMap(curX, curY, downLeft);
        continueRight = true;
    }
    curDirection = CurrentDirection.UP;
    curY--;
    spriteToUse = downPath;
}

private void GoLeft()
{
    // Рух вліво та оновлення карти
    UpdateMap(curX, curY, spriteToUse);
    curX--;
    spriteToUse = leftRight;
}

private void GoRight()
{
    // Рух вправо та оновлення карти
    UpdateMap(curX, curY, spriteToUse);
    curX++;
    spriteToUse = leftRight;
}
```

Рис. 3.18. методи GoUp, GoLeft і GoRight

У даній частині коду визначені методи GoUp, GoLeft і GoRight, що відповідають за рух шляху вгору, вліво і вправо відповідно.

Метод GoUp використовується для руху вгору. Залежно від поточного напрямку (ліво чи право), обирається відповідний спрайт (downRight або downLeft) для оновлення карти. Деякі становища, такі як continueLeft і continueRight, встановлюються для подальшого визначення напрямку шляху. Крім того, змінюється поточний напрямок на вгору, зменшується значення координати y, і встановлюється спрайт для використання в наступних кроках.

Методи GoLeft і GoRight відповідають за рух вліво та вправо відповідно. Вони викликають метод UpdateMap для оновлення карти з поточними координатами. Також вони встановлюють новий спрайт для використання та змінюють поточні координати x відповідно до напрямку руху.

Ці методи грають важливу роль у керуванні рухом шляху, забезпечуючи відповідний вигляд шляху в залежності від його напрямку та руху. Они є частиною стратегії генерації карт в грі або алгоритмів, де важлива логіка переміщення по та модифікації карти.

В наступній частині коду визначаються ряд функцій, що відповідають за оновлення інформації про плитку на карті в контексті генерації шляху або руху об'єктів на цій карті. Функція UpdateMap відповідає за зміну позначення плитки на карті, встановлення її координат та спрайту згідно з переданими параметрами. Зокрема, вона змінює координату по осі y, додаючи 0.3f до поточного значення, щоб створити візуальний ефект переміщення вгору на карті.

```
private void UpdateMap(int mapX, int mapY, Sprite spriteToUse)
{
    // Оновлення даних про плитку
    tileData[mapX, mapY].transform.position = new Vector2(tileData[mapX, mapY].transform.position.x, tileData[mapX, mapY].transform.position.y + 0.3f);
    tileData[mapX, mapY].tileID = 1;
    tileData[mapX, mapY].spriteRenderer.sprite = spriteToUse;
}
```

Рис. 3.19. оновлення даних плитки на карті

У цій частині коду визначаються функції для оновлення інформації про плитку на карті. Позначення нового положення плитки відбувається шляхом зміни її координати по осі y на 0.3f. Також присвоюється значення 1 полю tileID, що може слугувати індикатором наявності об'єкту на певній плитці. Змінюється спрайт плитки відповідно до значення spriteToUse, яке визначається в інших частинах коду. Ці дії реалізують механіку оновлення та візуального представлення картографії гри під час її виконання та розвитку шляху. Розглянемо кожний рядок окремо.

Рядок tileData[mapX, mapY].transform.position = new Vector2(tileData[mapX, mapY].transform.position.x, tileData[mapX, mapY].transform.position.y + 0.3f) визначає нове положення плитки на карті. Координата y збільшується на 0.3f, що може вказувати на підняття плитки вгору або віддалення її від гравітаційного центру гри.

В частині tileData[mapX, mapY].tileID = 1, поле tileID структури TileData встановлюється на 1. Ймовірно, це індикатор того, що плитка зайнята або щось знаходиться на цій позиції.

А за рахунок tileData[mapX, mapY].spriteRenderer.sprite = spriteToUse, спрайт для відповідної плитки оновлюється, і йому призначається значення spriteToUse, яке визначається в інших частинах коду, наприклад, при генерації шляху.

Ці рядки забезпечують відслідковування та зміну вигляду карти відповідно до подій та дій, які відбуваються під час генерації шляху у грі.

Ці функції взаємодіють з іншими частинами коду, такими як генерація карти (GenerateMap), вибір напрямку (ChooseDirection), та інші, для створення та зміни шляху в грі, а також візуального відображення цих змін на ігровому полі.

Отже розроблений код представляє собою Unity-додаток для генерації карт, що реалізує розроблений метод генерації рівнів для гри жанру Tower Defense. Основні елементи включають розміри та створення карти, префаб плитки та різні спрайти для шляху. Дані про кожну плитку зберігаються в структурі TileData, яка включає позицію, ідентифікатор та посилання на SpriteRenderer.

Ініціалізація відбувається в методі Awake(), де також ініціалізується масив tileData та генерується початкова частина карти. Основна логіка генерації шляху реалізована в методі GeneratePath(), де використовується ітератор для крокової генерації шляху, змінюючи напрямок та вибираючи новий напрямок при перешкодах.

CheckCurrentDirections() визначає можливість руху в поточному напрямку, змінюючи координати відповідно. Метод ChangeDirection() визначає логіку зміни напрямку руху, враховуючи випадкові значення та умови. Методи GoUp(), GoLeft() та GoRight() відповідають за рух вгору, вліво та вправо відповідно.

Оновлення даних про плитку відбувається в методі UpdateMap(), де змінюється позиція, ідентифікатор та спрайт плитки. Цей код намагається створити динамічний та непередбачуваний геймплей, забезпечуючи генеративний підхід до створення ігрових карт та лабіринтів.

3.5 Результати розробленого методу

Тестування розроблених методів чи програмних продуктів є невід'ємною складовою частиною дослідницьких процесів та інженерії програмного забезпечення. Це важлива фаза, спрямована на перевірку функціональності та відповідності розроблених рішень вимогам та очікуванням. Основні цілі тестування включають валідацію функціональності для перевірки коректності роботи методу чи програми відповідно до встановлених стандартів та технічних вимог.

Подальша мета полягає у підтвердженні відповідності результатів розробленого методу чи програмного продукту встановленим критеріям. Це включає перевірку правильності відтворення результатів відповідно до визначених вимог до функціональності та ефективності. Тестування допомагає виявити та усунути будь-які помилки чи недоліки, що можуть виникнути під час роботи програми, сприяючи стабільності та надійності розроблених рішень.

Нарешті, тестування спрямоване на підтвердження відповідності розробленого методу встановленим специфікаціям та вимогам. Це допомагає переконатися, що розроблені рішення відповідають технічним стандартам та можуть бути використані відповідно до заданих параметрів та очікувань. Такий аналіз та перевірка сприяють впевненості в правильності та готовності розробленого методу для практичного використання в різних областях та сценаріях застосування.

Отже, після того, як було створено додаток на основі розробленого методу генерації рівню необхідно перевірити, чи розроблена модель відповідає поставленим критеріям. Для цього, використовуючи додаток, було згенеровано 200 рівнів та отримано наступну статистику:

- 1 було згенеровано з розривом шляху;
- 1 не мало точок виходу чи входу;
- 3 згенерувалося без рельєфу;
- в 5 випадках шлях був занадто довгим.

Отримана статистика з генерації 200 рівнів застосованого методу генерації рівню має деякі ключові відмінності та недоліки, які варто врахувати.

Такі результати тестування свідчать про те, що розроблений метод має свої особливості та можливості для покращень, зокрема в аспектах безперервності шляхів, рельєфу та оптимізації генерації шляхів у складних моментах.

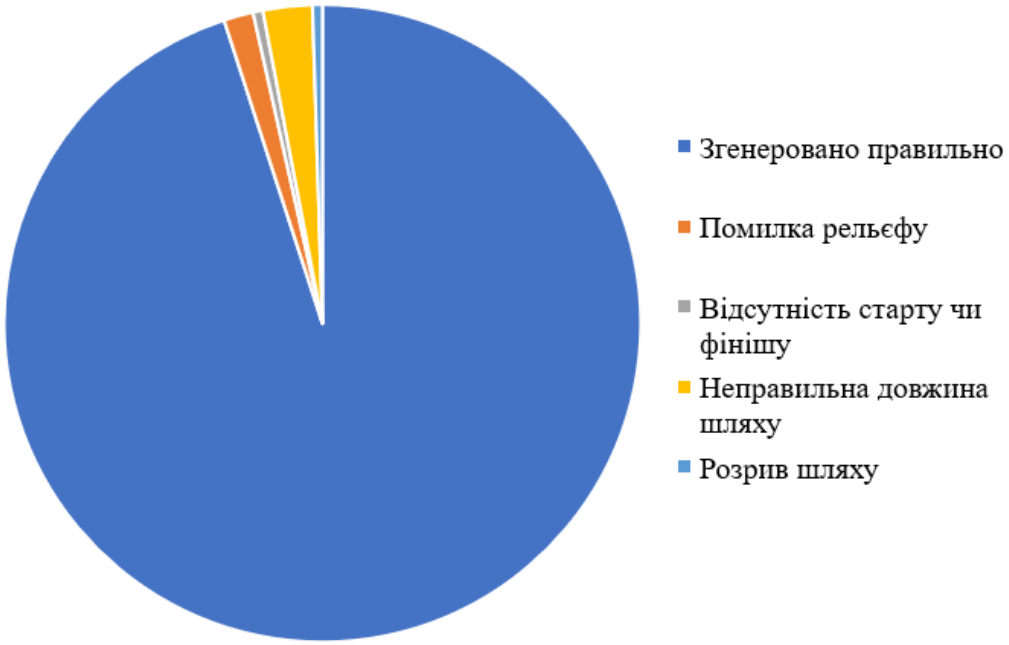


Рис. 3.20. Результати тестування розробленого методу

Для порівняння ефективності генерації також було протестовано інші поєднання для методів. Для прикладу розглянемо поєднання методу відхилення середини та алгоритму Дейкстри. При тій ж кількості генерацій він показав такі результати:

- 3 було згенеровано з розривом шляху;
- 2 не мало точок виходу чи входу;
- 6 згенерувалося без рельєфу;
- в 5 випадках шлях був занадто довгим або ж коротким.

Згідно отриманих результатів використання поєднання методу відхилення середини та алгоритму Дейкстри при генерації рівнів також показує свої особливості та недоліки.

Виявлення розриву шляху в трьох випадках може свідчити про певні недоліки в генерації безперервних шляхів у використовуваному методі. Також, виявлення в двох випадках відсутності точок входу чи виходу може вказувати на те, що це поєднання методів може не завжди гарантувати повноту створення шляхів у всіх рівнях.

Також виявлено шість випадків, коли рівні генерувалися без рельєфу, що може свідчити про обмежені можливості цього поєднання методів у створенні різноманітного ландшафту.

Наявність п'яти випадків, коли шлях був занадто довгим або коротким, може свідчити про потребу у вдосконаленні алгоритму генерації шляхів у цьому методі.

Отже, результати показують, що поєднання методу відхилення середини та алгоритму Дейкстри мають свої переваги, але також мають певні обмеження та недоліки, зокрема у формуванні безперервних шляхів, створенні рельєфу та управлінні довжиною шляху.

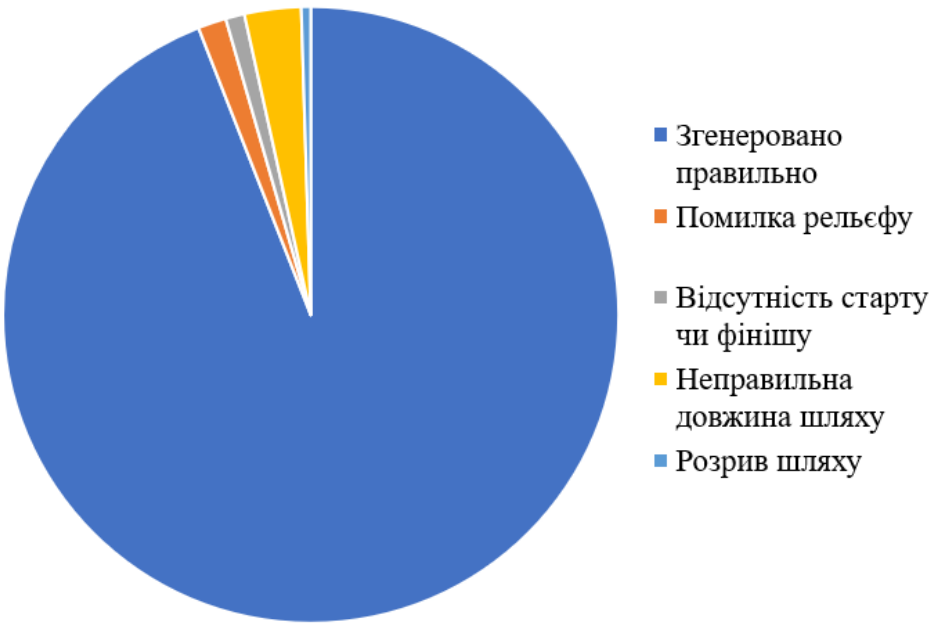


Рис. 3.21. Результати тестування методу поєднання відхилення середини та алгоритму Дейкстри

Інший метод, який є доссить популярним в генерації рівнів для гри в жанрі Tower Defense це поєднання генерації завдяки методу клітинного автомату та пошуку шляху за рахунок алгоритму колонії мурах. Такий метод показав наступні результати за ту ж саму кількість генерацій:

- 1 було згенеровано з розривом шляху;
- 7 не мало точок виходу чи входу;
- 8 згенерувалося без рельєфу;
- в 15 випадках шлях був занадто довгим або ж коротким.

Що говорить про те, шо метод, який використовує поєднання методу клітинного автомату та алгоритму колонії мурах для генерації рівнів у грі в жанрі Tower Defense, виявив деякі менш успішні характеристики під час генерації 200 рівнів.

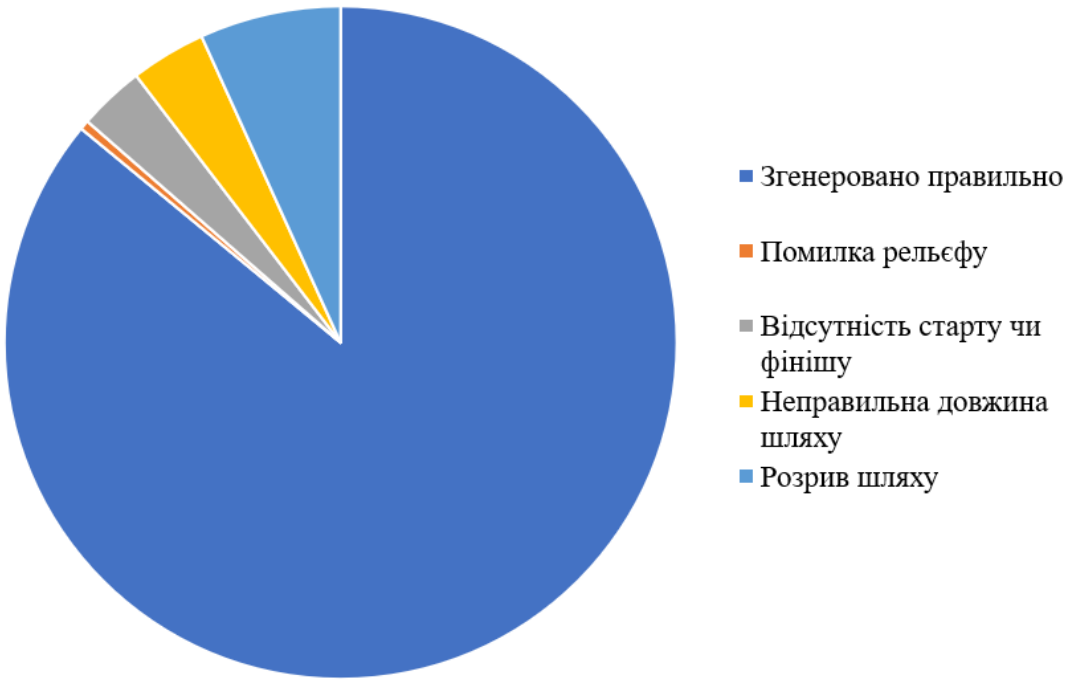


Рис. 3.22. Результати тестування методу поєднання генерації клітинним автоматом та пошуку шляху за рахунок алгоритму колонії мурах.

Хоча цей метод має лише один випадок з розривом шляху, він має високу кількість випадків, коли відсутні точки входу чи виходу -7 випадків, а також значну кількість рівнів без рельєфу- 8 разів. Однак особливо варто відзначити той факт,

що цей метод продемонстрував 15 випадків, коли шлях був занадто довгим або коротким.

Це свідчить про те, що використання алгоритму колонії мурах у поєднанні з методом клітинного автомату має свої обмеження та недоліки при генерації рівнів. Незважаючи на певні позитивні результати, такі як відсутність розривів у шляху, велика кількість помилок у відсутності входу або виходу, а також проблеми з довжиною шляху, можуть вплинути на якість рівнів.

Порівнюючи отримані результати генерації рівнів для гри в жанрі Tower Defense за допомогою трьох різних методів, можна зробити кілька спостережень.

Розроблений метод, що використовує поєднання Перліна та алгоритму A*, показав стабільні результати. З всього набору рівнів лише один мав розрив у шляху, і лише в 3 випадках було згенеровано рівні без рельєфу. Проте 5 випадків, коли шлях був занадто довгим, може вказувати на певні труднощі з оптимізацією.

Поєднання методу відхилення середини та алгоритму Дейкстри продемонструвало схожу кількість розривів у шляху, але виявило більшу кількість випадків, коли шлях був занадто довгим або коротким. Це може свідчити про обмеженість точності та ефективності цього підходу в генерації більш різноманітних та збалансованих рівнів.

Нарешті, поєднання методу клітинного автомату та алгоритму колонії мурах показало значну кількість випадків, коли шлях був занадто довгим або коротким, а також велику кількість випадків, коли не було виходу або входу на рівні. Це свідчить про проблеми з точністю генерації шляху та його структури.

Отже, порівнюючи ці три методи генерації рівнів, поєднання Перліна та алгоритму A* виявилося більш стабільним та збалансованим за загальною кількістю помилок, що дозволяє стверджувати, що цей метод кращий за вже існуючі методи створення рівнів для ігор жанру Tower Defense.

Також отримані результати можна сформувати у вигляді таблиці.

Таблиця 3.1

Порівняльна таблиця результатів тестування методів

Метод	Кількість правильних генерацій	Кількість генерацій які не задовільнили критерії	Відсоток не правильно згенерованих рівнів
Розроблений метод	190	10	5%
Поєднання методу відхилення середини та алгоритму Дейкстри	184	16	8%
Поєднання генерації клітинним автоматом та пошуку шляху за рахунок алгоритму колонії мурах	169	31	16%

На основі отриманих результатів тестування різних методів генерації рівнів для гри в жанрі Tower Defense можна зробити кілька висновків.

По-перше, розроблений метод, що використовує комбінацію методу Перліна та алгоритму A*, продемонстрував найкращу ефективність серед випробуваних. З 200 генерацій лише 10 не відповідали критеріям, що становить 5% від загальної кількості. Це свідчить про стабільність та високу точність цього методу.

По-друге, поєднання методу відхилення середини та алгоритму Дейкстри показало меншу ефективність порівняно з розробленим методом. З 16% неправильно згенерованих рівнів, цей метод виявився менш стабільним та мав більше недоліків, що може вказувати на меншу точність у формуванні шляхів.

Нарешті, поєднання генерації клітинним автоматом та алгоритму колонії мурах продемонструвало найнижчу ефективність серед усіх методів. З 31% неправильно згенерованих рівнів, цей метод виявився менш точним та стабільним, що може ускладнити його використання у грі.

ВИСНОВКИ

Після проведеного аналізу та досліджень магістерської роботи можна зробити такі висновки.

Було розглянуто головні підходи для реалізації генерації випадкових рівнів для гри в жанрі Tower Defense, та виявлені головні елементи притаманні цьому жанру та особливостям побудови карти.

Проаналізовано методи створення рівнів цього жанру та виявлено переваги та недоліки існуючих підходів генерації рівнів. Це дозволило визначити доцільність використання методу шумів Перліна для генерації мапи та A* алгоритму для прокладання оптимального шляху в грі.

Розроблена модель, що використовує метод шумів Перліну для генерації мапи та A* алгоритм для створення маршруту, що показала відмінні результати. Реалізація цієї моделі у вигляді програмного забезпечення дозволяє автоматизувати процес генерації рівнів для ігор жанру Tower Defense.

Тестування розробленого рівня показало, що лише 5% рівнів містять помилки, що вказує на високу ефективність та точність розробленої моделі. Таким чином, цей дослідницький проект підтвердив придатність та відповідність розробленого методу для генерації рівнів у грі жанру Tower Defense, використовуючи метод шумів Перліну та A* алгоритм для прокладання оптимального маршруту в грі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Y. Du et al., "Automatic level Generation for Tower Defense Games," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 2019, pp. 670-676, doi: 10.1109/ITNEC.2019.8728989.
2. Øystein Brox, Procedural Generation of Terrain and Roads, Master's thesis in Cybernetics and Robotics, June 2019, 106 p.
3. Simon Liu, Li Chaoran, Li Yue, Ma Heng, Hou Xiao, Shen Yiming, Wang Licong, Chen Ze, Guo Xianghao, Lu Hengtong, Du Yu, and Tang Qinting. 2019. Automatic generation of tower defense levels using PCG. In Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG '19). Association for Computing Machinery, New York, NY, USA, Article 10, 1–9. https://doi.org/10.1145/3337722.3337723
4. Anirma Kandida Ginting, Kartika Sari, Cut Fadhilah, Rahmad Nurhadi Yusra, Dedy Hartama and Muhammad Zarlis, "Application of the Perlin Noise Algorithm as a Track Generator in the Endless Runner Genre Game", Journal of Physics: Conference Series, Volume 1255, The International Conference on Computer Science and Applied Mathematic 10–12 October 2019, Niagara Hotel, Parapat, Indonesia, 5p.
5. Asfarian, A., Ramadhan, W., Putra, W. A., Raharjanto, G. M., & Frisky, R. (2019). Creating a Circular Tower Defense Game: Development and Game Experience Measurement of Orbital Defense X. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, 4(3), 249-258. https://doi.org/10.22219/kinetik.v4i3.779
6. Johan Öhman. 2020. Procedural Generation of Tower Defense levels. http://www.diva-portal.org/smash/get/diva2:1442180/FULLTEXT01.pdf
7. Groeschel, M., & SchÃ¤fer, T. (2020). Analysis of Mobile App Revenue Models Used in the most Popular Games of the Tower Defense Genre on Google Play. Asian Journal of Computer and Information Systems, 8(1). https://doi.org/10.24203/ajcis.v8i1.6038
8. Hubbel, Gaines S. What Is A Game. 3rd Edition. Jefferson: Mc Farland & Company Inc., 2020, 219p.
9. Risi, S., Preuss, M. From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI. Künstl Intell 34, 7–17 (2020). https://doi.org/10.1007/s13218-020-00647-w
10. Dhinakar Gogi, Aditya Jadhav, Nayan Shingare, Joel Thomas, & Dakshayani G. (2021). Strategic Tower Defence Game. International Journal of

Engineering and Management Research, 11(3), 73–77. https://doi.org/10.31033/ijemr.11.3.12

11. Zafar, A., Mujtaba, H., & Beg, O. (2021). Procedural Content Generation for General Video Game Level Generation. Inteligencia Artificial, 24(68), 33–36. https://doi.org/10.4114/intartif.vol24iss68pp33-36

12. Procedural Content Generation for Tower Defense Games: Preliminary Experiment with Reinforcement Learning. Yueming Xu Tetsuro Tanaka. The 26th Game Programming Workshop 2021

13. Kraner, V., Fister, I., Brezočnik, L. (2021). Procedural Content Generation of Custom Tower Defense Game Using Genetic Algorithms. In: Karabegović, I. (eds) New Technologies, Development and Application IV. NT 2021. Lecture Notes in Networks and Systems, vol 233. Springer, Cham. https://doi.org/10.1007/978-3-030-75275-0_54

14. Desando Anugrah Ramadhan, Aries Dwi Indriyanti. Procedural Content Generation pada Game World Exploration Sandbox Menggunakan Alogoritma Perlin Noise. In: Journal of Informatics and Computer Science): Volume 04 Nomor01, 2022(JINACS), pp. 86-91 doi: https://doi.org/10.26740/jinacs.v4n01.p86-91

15. Ken Perlin.Chapter 4.In the beginning: The Pixel Stream Editor. 12p. [електронний ресурс] URL: https://redirect.cs.umbc.edu/~olano/s2002c36/ch04.pdf (дата звернення 23.10.2023)

16. Diogo Miguel P.L. Dias, Joao Paulo P. Sousa, Barbara C.V.B. Barroso, Ines M.B. Magalhaes. A Novel Procedural Content Generation Algorithm for Tower Defense Games. ICACS '22: Proceedings of the 6th International Conference on Algorithms, Computing and SystemsSeptember Article No: 9, 2022 pp 1–7 https://doi.org/10.1145/3564982.3564993

17. R. C. E. Silva, N. Fachada, D. De Andrade and N. Códices, "Procedural Generation of 3D Maps With Snappable Meshes," in IEEE Access, vol. 10, pp. 43093-43111, 2022, doi: 10.1109/ACCESS.2022.3168832.

18. Hans Juwiantho, Liliana Liliana, Michael Budiono. “Procedural Content Generation pada Game Tower Defense menggunakan Perlin Noise dan Algoritma Floyd Warshall” in Journal of Animation and Games Studies (JAGS), Vol 9, No 1, pp.11-28, 2023, DOI: https://doi.org/10.24821/jags.v9i1.8100

19. John Xu, John Morris. “Procedural generation in 2d metroidvania game with answer set programming and perlin noise” in International Journal of Artificial Intelligence and Applications (IJAIA), Vol. 14, No.3, pp. 21- 21-35, May 2023

20. Ahmed , S., & Pandey, B. (2023). Procedural Terrain Generation by Sampling a 2D Monochrom Perlin Noise Map in Unity. Asian Journal of Research in Computer Science, 16(1), 37–42. https://doi.org/10.9734/ajrcos/2023/v16i1333

21. 2D skeleton - isometric pixelart character | 2D characters | unity asset store. Unity Asset Store - The Best Assets for Game Making. URL: https://assetstore.unity.com/packages/2d/characters/2d-skeleton-isometric-pixelart-character-254596 (дата звернення: 24.11.2023).

22. Isometric tower defense pack | 2D environments | unity asset store. Unity Asset Store - The Best Assets for Game Making. URL: https://assetstore.unity.com/packages/2d/environments/isometric-tower-defense-pack-183472 (дата звернення: 01.12.2023).

23. Unity - manual: unity user manual 2022.3 (LTS). Unity - Manual: Unity User Manual 2022.3 (LTS). URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 30.11.2023).

24. Visual Studio documentation. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022 (дата звернення: 30.11.2023).

25. Red Blob Games: making maps with noise. Red Blob Games. URL: https://www.redblobgames.com/maps/terrain-from-noise/ (дата звернення: 30.11.2023).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

МАГІСТЕРСЬКА РОБОТА

«РОЗРОБКА МЕТОДУ ВИПАДКОВОЇ ГЕНЕРАЦІЇ РІВНІВ ДЛЯ ГРИ В ЖАНРІ TOWER DEFENSE»

Виконала: студентка групи ПДМ-64, Ігнатова Марія Володимирівна

Керівник: доктор філософії, доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ - 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження: підвищення якості створення рівнів в іграх жанру Tower Defense за рахунок використання автоматичної генерації.

Об'єкт дослідження: процес генерації випадкових рівнів в грі жанру Tower Defense.

Предмет дослідження: алгоритми штучного інтелекту для генерації рівнів.

КЛЮЧОВІ ЕЛЕМЕНТИ РІВНІВ ДЛЯ ГРИ В ЖАНРІTOWER DEFENSE

- карта
- шлях
- вежі
- супротивники
- система нагород

3

ІСНУЮЧІ МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ СТВОРЕННЯ РІВНІВ ДЛЯ ГРИ В ЖАНРІ TOWER DEFENSE

Метод	Переваги	Недоліки
Ручне створення	• Надійне	• Довгий час створення • Витрата великої кількості ресурсів
Процедурне генерування	• Швидше у виготовлені • Дешевше в ресурсах • Більш продуктивне	• Видає помилки

4

ПРОЦЕС СТВОРЕННЯ РІВНЯ В ГРІ ЖАНРУ TOWER DEFENSE

5

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА МЕТОДІВ ГЕНЕРАЦІЇ МАПИ ДЛЯ РІВНЯ

Метод	Переваги	Недоліки
Метод відхилення середини	Рекурсивна природа алгоритму дозволяє легко створювати деталізовані елементи ландшафту	Можливі помилки в генерації Низька точність
Метод генерації за допомогою клітинного автомату	Проста реалізація Низька складність коду та вимоги до ресурсів	Деякі конфігурації клітинних автоматів можуть призводити до нерівномірності або недосконалості генерованого ландшафту Не збалансовані
Алгоритм «Квадрат-ромб»	Швидкість	Складність реалізації
Метод шуму Перліна	Гладкість та безперервність Параметризована генерація Проста реалізація	Чутливість до параметрів Складність використання на великих масштабах

6

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА МЕТОДІВ ПОШУКУ ШЛЯХУ

Метод	Переваги	Недоліки
Алгоритм A*	Завжди знайде рішення і найкоротши шлях; Зосереджується лише на цільовому вузлі; Розширює найменшу кількість вузлів, щоб знайти оптимальний шлях. Надійність	Складний у виконанні; Не оптимально, коли існує декілька ціл Споживає багато пам'яті; Має проблеми зі складністю, оскільки швидкість виконання сильно залежить в точності евристики.
Алгоритм Дейкстри	можна використовувати для вирішенн відносно великих проблем використовується в маршрутизації, роботизованому виявленні шляху, файловому сервері тощо Низька складність,оптимальний та рівномірний Майже лінійний і працює як для спрямованого, так і для неорієнтован графа.	Витрачайте час на сліпий пошук Не може впоратися з негативними края Приводить до ациклічних графів
Генетичний алгоритм	Висока швидкість конвергенції.	Легко впасти в передчасну конвергенці Залежить від початкової популяції.
Техніка оптимізації колонії мурах	Проста реалізація; Легко розпаралелюється для одночас обробки; Без похідних.	Розподіл ймовірностей змінюється на ітерації; Час для конвергенції невизначений.

7

МАТЕМАТИЧНА МОДЕЛЬ МЕТОДУ ГЕНЕРАЦІЇ МАПИ

Формула для обчислення інтерполяції шуму Перліна

$$P = (x, y), i = floor(x), j = floor(y) \quad (1)$$

$$\overrightarrow{g_{00}} = grad(i, j), \overrightarrow{g_{10}} = grad\,(i + 1, j) \quad (2)$$

$$\overrightarrow{g_{01}} = grad(i, j + 1), \overrightarrow{g_{11}} = grad(i + 1, j + 1) \quad (3)$$

$$u = x - i, v = y - j \quad (4)$$

$$n_{00} = \overrightarrow{g_{00}} \cdot \begin{bmatrix} u \\ v \end{bmatrix}, n_{10} = \overrightarrow{g_{10}} \cdot \begin{bmatrix} u - 1 \\ v \end{bmatrix}, \quad (5)$$

$$n_{01} = \overrightarrow{g_{01}} \cdot \begin{bmatrix} u \\ v - 1 \end{bmatrix}, n_{11} = \overrightarrow{g_{11}} \cdot \begin{bmatrix} u - 1 \\ v - 1 \end{bmatrix},$$

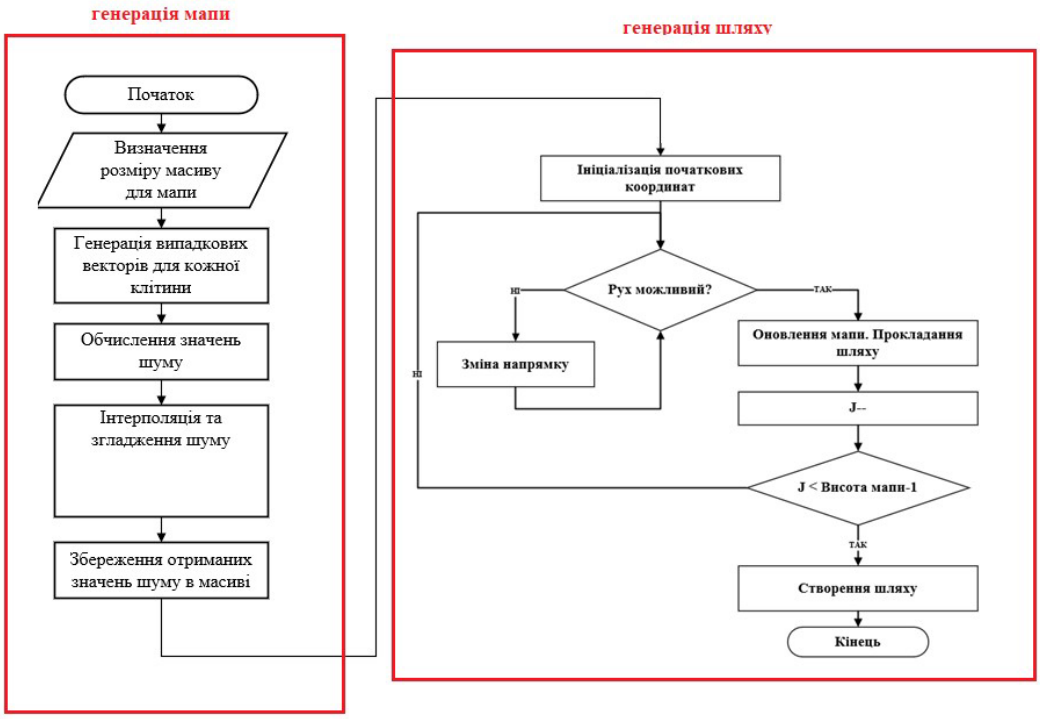
$$n_{x0} = n_{00}\big(1 - f(u)\big) + n_{10} f(u), n_{x1} = n_{01}\big(1 - f(u)\big) + n_{11} f(u) \quad (6)$$

$$n_{xy} = n_{x0}\big(1 - f(v)\big) + n_{x1} f(v), \quad (7)$$

де:
P=(x,y) представляє координати точки у просторі;
i=floor(x) та j=floor(y) є цілими частинами координат x та y відповідно;
$g00$,$g10$,$g01$,$g11$ - це градієнти шуму для чотирьох сусідніх вершин;
v=y–j представляють різниці між координатами x та y і цілими частинами i та j відповідно;
$n00$,$n10$,$n01$,$n11$ - скалярні добутки між градієнтами та відповідними відстанями між (x,y) та кожною з вершин сітки;
$nx0$,$nx1$,nxy - результати лінійної інтерполяції між $n00$,$n10$ та $n01$,$n11$ для x-координати та y-координати відповідно.

8

АЛГОРИТМ ГЕНЕРАЦІЇ РІВНЯ



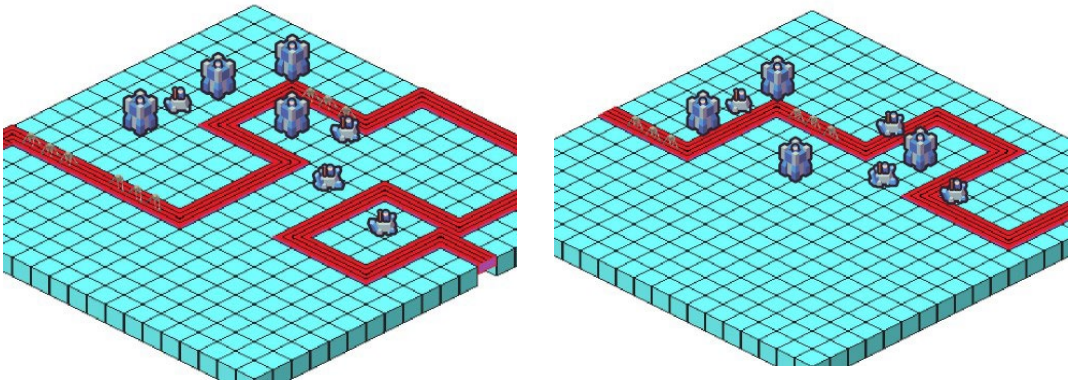
9

КРИТЕРІЇ ГЕНЕРАЦІЇ РІВНЮ

Критерій	Позначення	Значення якого може набути	Як вимірюється	Значення змінних
Координати початку та кінця шляху	W	1 0	$W = \begin{cases} 1, \text{якщо в шляху є точка входу та виходу,} \\ 0, \text{якщо бодай одна з них відсутня.} \end{cases}$,	де: W – Критерій початкових координат;
Відсутність розривів	C	1 0	$C = \begin{cases} 1, \text{якщо } G = 0 \\ 0, \text{ якщо } G > 0; \end{cases}$	де: C – значення критерію розривів. $G = G_{start} + n,$ де: G – сума розривів; G_{start} – початкове значення кількості розривів, яке рівне 0; n – кількість розривів шляху.
Наявність рельєфу	R	1 0	$R = \begin{cases} 1, \text{якщо рельєф є ,} \\ 0, \text{якщо рельєф не сформований.} \end{cases}$	де: R – Результат критерію наявності рельєфу;
Довжина шляху	L	1 0	$L = \begin{cases} 1, \text{якщо } l < \frac{n}{2} \text{ і } l > h \\ 0, \text{інакше} \end{cases}$,	де: L - критерій довжини шляху; l – довжина шляху; n – кількість клітинок мапи; h - довжина мапи.

10

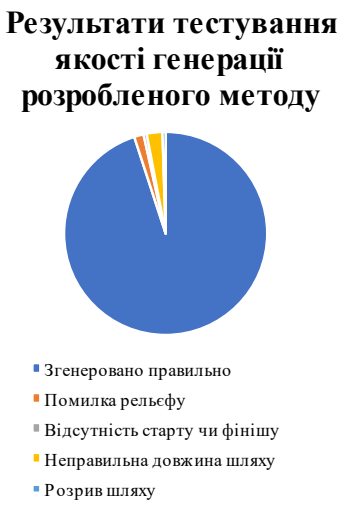
ПРИКЛАД ГЕНЕРАЦІЇ РІВНЯ



11

РЕЗУЛЬТАТИ ТЕСТУВАННЯ РІВНІВ

Метод	Кількість правильних генерацій	Кількість генерацій які не задовільнили критерії	Відсоток не правильно згенерованих рівнів
Розроблений метод	190	10	5%
Поєднання методу відхилення середини та алгоритмуДейкстри	184	16	8%
Поєднання генерації клітинним автоматом та пошукушляхуза рахунок алгоритму колонії мурах	169	31	16%



12

ВИСНОВКИ

1. Розглянуто головні підходи для реалізації генерації випадкових рівнів для гри в жанрі Tower Defense, та виявлені головні елементи притаманні цьому жанру та особливостям побудови карти.
2. Проведено аналіз існуючих методів, що використовуються для випадкового створення рівнів ігор жанру Tower Defense. Виявлено переваги та недоліки особливостей створення генерації рівнів. Вирішено доречним використовувати в розробці методу такі підходи як генерацію мапи з використанням методу шумів Перліну, та прокладання шляху за рахунок A* алгоритму.
3. Визначено критерії якості, за допомогою яких проведено аналіз характеристик створених рівнів.
4. Розроблена модель генерації рівнів на основі шумів Перліна та A* алгоритму для пошуку шляху, що прокладається для пересування ворогів
5. Розроблено додаток, який реалізує автоматизовану генерацію випадкових рівнів на основі шумів Перліну та A* алгоритму для гри в жанрі Tower Defense. Проведено моделювання шляхом генерації ігрових рівнів.
6. Проведено тестування створених рівнів за результатами яких відсоток помилково згенерованих становить 5%.

АПРОБАЦІЯ

Тези:

1. Ігнатова М.В., Дібрівний О.А. Огляд методів створення рівнів для ігор жанру tower defense. IV Проблеми комп'ютерної інженерії : Науково-практ. конф., м. Київ, 1 груд. 2023 р. Київ, 2023. 133-134с.
2. Ігнатова М.В., Дібрівний О.А. Генерація рівнів tower defense за рахунок шуму перліна. IV Проблеми комп'ютерної інженерії : Науково-практ. конф., м. Київ, 1 груд. 2023 р. Київ, 2023 131-132с.

Стаття:

1. Ігнатова М.В., Дібрівний О. А. Створення рівнів для гри в жанрі tower defense. Зв'язок. (подано в редакцію, очікує рецензії)

ДЯКУЮ ЗА УВАГУ!