МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

**ПАРХОМЕНКО ВЛАДИСЛАВ ВАДИМОВИЧ**

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____ О.В. Зелінська
«_____» _____ 2024 р.

# ІГРОВИЙ ЗАСТОСУНОК ЖАНРУ ROGUELIKE З ГЕНЕРАЦІЄЮ ВИПАДКОВИХ ПОДІЙ

Спеціальність 122 Комп’ютерні науки

**Кваліфікаційна (бакалаврська) робота**

Керівник:
Потапова Н. А., доцент кафедри
інформаційних технологій,
(назва кафедри)
канд. екон. наук, доцент

Оцінка: _____/_____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2024

## АНОТАЦІЯ

**Пархоменко В.В. Ігровий застосунок жанру Roguelike з генерацією випадкових подій.** Спеціальність 122 «Комп'ютерні науки», Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі досліджено хронологію змін підходів у створенні ігрових застосунків та оцінку різних підходів генерації випадкових подій у жанрі "Roguelike". Показано сутність використання сучасних алгоритмів генерування випадкових подій, та їх вплив на досвід користувача. Обгрунтовується вплив генерування випадкових подій на підвищення інтересу користувачів, створюючи унікальний досвід та нові виклики у кожній наступній ігровій сесії.

**Ключові слова**: рогалик, рандом, випадкові події, реіграбельність, ігровий досвід.

57 с., 22 рис., 40 джерел.

## ABSTRACT

**Parkhomenko V.V. Game application of the genre Roguelike with the generation of random events.** Specialty 122 "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

The qualification (bachelor's) thesis examines the chronology of changes in approaches to creating game applications and evaluates different approaches to generating random events in the Roguelike genre. The essence of using modern random event generation algorithms and their impact on user experience is shown. The impact of random event generation on increasing user interest by creating a unique experience and new challenges in each subsequent game session is substantiated.

**Keywords:** roguelike, random, random events, replayability, gaming experience.

57 p., 22 figures, 40 sources.

# ЗМІСТ

# ВСТУП

Відеоігри є одним із найцікавіших здобутків сучасних інформаційних технологій, участь в який є не тільки захоплюючою мандрівкою в уявлений світ, а й покращення психологічного стану. Різновиди жанрів відеоігор розкрили широкий спектр нових напрямів розвитку ігрової індустрії. Відеоігри, особливо такі жанри, як рогалики, процвітають завдяки залученню гравців і створенню відчуття постійного виклику. Реалізація даної концепції вимагає зосередження на розвитку наступних ключових аспектів проблематики геймінгу рогаликів:

1. Висока реіграбельність: рогалики відомі своїми механіками перманентної смерті та процедурно згенерованими рівнями. Це означає, що кожне проходження гри пропонує унікальний досвід. Випадкові події ще більше посилюють цей ефект, створюючи непередбачувані ситуації та змушуючи гравців адаптувати свої стратегії.

2. Глибше залучення гравців: випадкові події створюють моменти несподіванки і змушують гравців приймати швидкі рішення з потенційно високими ставками. Така активна участь утримує гравців у грі та створює додатковий рівень захоплення, який виходить за рамки простого проходження рівнів.

Об’єднавши у собі низку цікавих дизайнерських ідей та ігрових моделей випадкових подій, розробки ігор жанру рогаликів залишаються актуальним і перспективним напрямом.

*Метою* кваліфікаційної (бакалаврської) роботи є створення ігрового додатку, який вирізняється динамічним ігровим процесом, що підживлюється системою генерування випадкових подій.

*Об’єктом дослідження* є процес розробки ігрового додатку в жанрі рольової гри, який охоплює такі етапи як-от: ігровий дизайн, програмування та візуальну частину.

*Предметом дослідження* є система генерації випадкових подій в рамках рольової гри, де основний фокус зміщений до основної інновації проєкту –

дизайну подій.

Основними завданнями, які вирішувались у роботі є:

1. Провести аналіз теоретичних основ створення ігор жанру рогаликів з акцентуванням уваги на історичну хронологію еволюційних подій розвитку даних ігор та можливості їх розробок у майбутньому.

2. Виявити основні особливості категорії випадковості в ігрових застосунках, її вплив на формування ідеології ігор та набуття досвіду гравців.

3. Дослідити основні тенденції, що склались на ринку ігрової індустрії та оцінити сучасні алгоритмічні інновації з потенційними недоліками ігор рогаликів.

4. Сформувати основні механіки гри та класифікувати основні види випадкових подій в межах проходження гри.

5. Розробити ігровий застосунок жанру рогалик із використанням алгоритму генерації випадкових подій.

6. Реалізувати програмні модулі ігрового застосунку та провести тестову демонстрацію розробленої гри.

*Практичне значення* роботи полягає у відтворенні гри, яка здатна реалізувати рольову гру на основі аналізу взаємодії гравця з реалізованими випадковими подіями.

*Структура роботи* складається із вступу, трьох основних розділів, висновків, переліку використаних посилань та додатків. Перший розділ основної частини роботи присвячений історії розвитку жанру «Roguelike», сучасним представникам та можливостям в реалізації ігроладу. В другому розділі зосереджена увага на підборі ключових алгоритмів і методів створення застосунку. Третій розділ демонструє практичну реалізацію розроблених інтерфейсу та застосунку в цілому.

Апробація результатів дослідження представлена на III всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Прикладні інформаційні технології» (м. Вінниця 7 липня 2022 року) у доповіді: Пархоменко В.В. Потапова Н.А. «Використання технології Nanite в UE5».

# РОЗДІЛ 1.
# ТЕОРЕТИЧНІ ОСНОВИ ЖАНРУ ROGUELIKE

## 1.1 Історія та еволюція жанру Roguelike

Жанр Roguelike, відомий своєю високою складністю, процедурно згенерованими підземеллями та перманентною смертю, зайняв унікальну нішу в ігровому світі. Але його витоки є набагато глибшими, ніж здається, і сягають корінням від ранніх комп'ютерних технологій та геймерської любові до досліджень і викликів.

Хоча гра 1980 року «Rogue» дала назву жанру, насіння було посіяне кількома роками раніше [1]. У 1970-х роках з'явилися текстові пригодницькі ігри, які мали вирішальний вплив на рогалики. Ці ігри, в які часто грали на комп'ютерах в університетах або на ранніх домашніх комп'ютерах, покладалися на уяву гравців, щоб заповнити візуальні прогалини. Серед них виділяються дві ключові назви:

1) Colossal Cave Adventure (1975): Ця новаторська гра, також відома як «Adventure», представила концепцію дослідження віртуального світу за допомогою текстових описів. Гравці вводили команди, щоб взаємодіяти з навколишнім середовищем, розв'язувати головоломки та зустрічати фантастичних істот. Хоча гра не була бродилкою у строгому сенсі, вона продемонструвала можливості інтерактивного оповідання та дослідження у цифровому світі.


Рисунок 1.1 - Ігровий процес «Colossal Cave Adventure»

2) «Beneath Apple Manor» (1978): Вважається першою комерційною рольовою грою, яка запропонувала чітке дерево подій того, що мало статися в майбутньому. Розроблена для Apple II, Beneath Apple Manor представляла собою бродилку в підземеллі від першої особи з покроковим боєм, випадково згенерованими рівнями та перманентною смертю. За сучасними мірками, гра була недопрацьованою, але вона заклала основні механіки, які стали визначальними для жанру.


Рисунок 1.2 – Ігровий процес «Beneath Apple Manor»

1980 рік ознаменувався переломним моментом в історії ігор з виходом гри «Rogue». Розроблена Майклом Тоєм та Гленом Вічманом для Unix-систем, Rogue не була першою грою, в якій було використано просування підземеллями, процедурну генерацію чи перманентну смерть. Однак вона стала каталізатором нового жанру, назавжди закарбувавши своє ім’я в історії ігор. [2-4]

Кінець 1970-х - початок 1980-х років був часом, коли графічні можливості персональних комп’ютерів все ще були обмеженими. У цьому середовищі процвітали текстові пригодницькі ігри, що пропонували гравцям світ уяви, підживлений описовою прозою. «Rogue» успадкувала цю спадщину, використовуючи культову естетику ASCII. Гравці мандрували підземеллями, позначеними такими символами, як «@» (гравець), «W» (стіна) та «>» (сходи). Візуально простий, цей формат дозволяв деталізувати підземелля та розпалював уяву гравців.

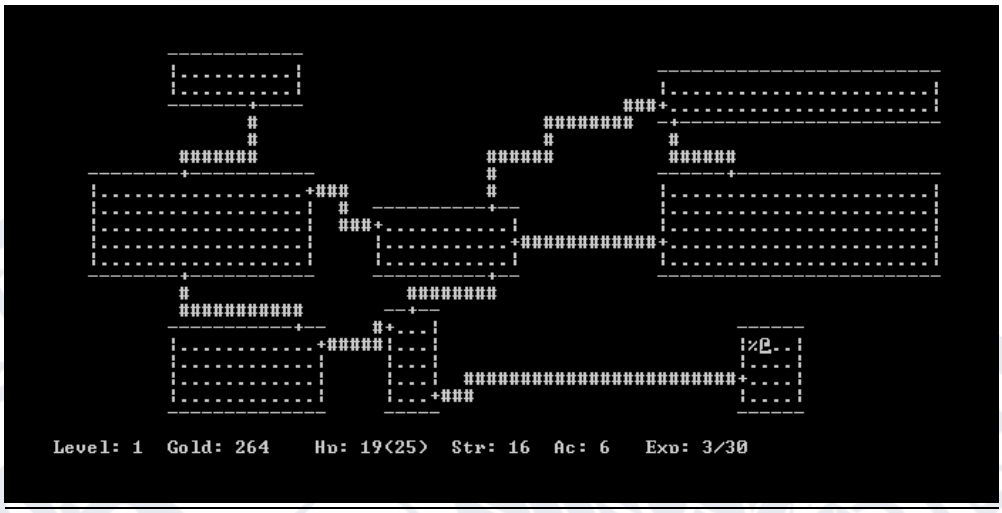


Рисунок 1.3 – Ігровий процес «Rogue»

Одним з найважливіших внесків «Rogue» було використання процедурної генерації. На відміну від попередніх ігор зі статичними підземеллями, рівні «Rogue» будувалися випадковим чином при кожному проходженні. Це нововведення додало непередбачуваності, яка захопила гравців. Жодне з двох проходжень не було однаковим, що заохочувало дослідження та адаптацію. Важливо, що вихідний код «Rogue» був у вільному доступі. Це сприяло процвітанню спільноти програмістів-любителів, які доопрацьовували, модифікували та розвивали основну механіку. З'явилися такі ігри, як «Hack» (фентезі-версія «Rogue») та «Moria» (відома завдяки своєму зануренню в сетінг Володаря Перснів), які розширили межі жанру. Сам термін «roguelike» почав з'являтися в групах новин Usenet у цей період, закріплюючи ідентичність жанру.

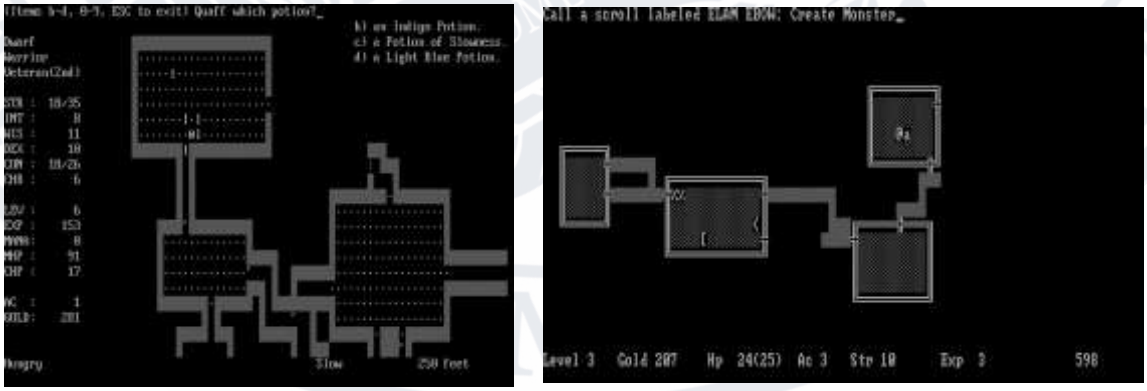

Рисунок 1.4 – Ігровий процес "Hack" та "Moria"

У новому тисячолітті спостерігався сплеск розвитку рольових ігор. Такі

ігри, як «NetHack» (складна і довговічна класика) та «Dungeon Crawl Stone Soup» (шедевр, розроблений спільнотою) продовжували вдосконалювати основну формулу. Однак почали з'являтися й інновації. Такі ігри, як «Shiren the Wanderer», запровадили систему руху на основі сітки, а «Brogue» запропонувала мінімалістичну естетику з дивовижною глибиною.

У період розквіту рогаликів, що охоплює кінець 20-го століття і закінчуючи сьогоденням, цей жанр пережив сплеск популярності та інновацій. Почавшись з ранніх текстових бродилок у підземеллях, таких як «Rogue» (1980), рогалики перетворилися на різноманітні та складні ігрові процеси на різних платформах. Ця еволюція була зумовлена розвитком технологій, принципів геймдизайну та креативності розробників. [5-8]

Однією з визначальних особливостей рогаликів є їхні процедурно згенеровані рівні, які гарантують, що жодне з двох проходжень не буде однаковим. Такий динамічний дизайн рівнів не лише забезпечує нескінченну реіграбельність, але й змушує гравців адаптуватися до постійно мінливого оточення та сценаріїв. Включення механіки перманентної смерті, коли персонаж гравця безповоротно помирає після поразки, додає рівня напруженості та прийняття рішень щодо ризику та винагороди, що є центральним елементом гри в жанрі roguelike.[9]

Випадкова генерація подій ще більше підвищує непередбачуваність рогаликів, вводячи широкий спектр несподіваних викликів, можливостей і сюжетних поворотів. Ці події можуть приймати різні форми, наприклад, зустріч з могутніми монстрами, відкриття захованих скарбів або зіткнення з екологічною небезпекою. Випадковість цих подій гарантує, що гравці повинні постійно бути напоготові, розробляючи стратегію та імпровізуючи у відповідь на обставини, що складаються.

Більше того, розквіт рогаликів приніс процвітання інді-розробці ігор, коли невеликі студії та окремі розробники пропонували інноваційні ідеї та розширювали межі жанру. Така демократизація інструментів і платформ для розробки ігор призвела до поширення ігор у жанрі рогаликів на ПК, консолях і

мобільних платформах, які задовольняють різноманітне коло гравців.

Інді-розробники, зокрема, стали синонімом інновацій у жанрі roguelike. Їхні менші масштаби дозволяють ризикувати та експериментувати, що призводить до постійного потоку винахідливих ігор, які постійно переосмислюють те, чим може бути рольова гра. Незалежно від того, чи це поєднання механіки рольової гри з платформером, стратегією або навіть з наративною історією, інді-розробники розширюють межі жанру, зберігаючи його свіжим і захопливим.

Втім, вплив принципів дизайну рольових ігор виходить за межі інді-сцени. Мейнстрім розробники та видавці визнали привабливість перманентної смерті, процедурної генерації та емерджентного сторітелінгу, інтегруючи ці елементи у франшизи блокбастерів та ААА-ігор. Таке мейнстрімне прийняття продемонструвало механіки рогаликів ширшій аудиторії, закріпивши їхнє місце в ігровій культурі.[10-12]

Більше того, еволюція roguelike-механіки не припиняється. Алгоритми генерації процедур стають все більш досконалими, постійно створюючи мінливі світи, які здаються одночасно складними та справедливими. Системи прогресії пропонують осмислений вибір і кастомізацію, гарантуючи, що кожен запуск буде унікальним і корисним. Інтеграція розповіді забезпечує контекст і глибину подорожі гравця, занурюючи його в продуманий ігровий світ.[13-15]

Проте, саме спільнота по-справжньому вдихає життя в цей жанр. На форумах, і в соціальних мережах вирують дискусії, обмін стратегіями та останніми новинами про рольові ігри. Спільноти модераторів процвітають, пропонуючи гравцям можливість пристосувати свій досвід до власних уподобань і розширити існуючі ігри.

По суті, жанр рольових ігор являє собою динамічний перетин творчості, виклику та ком’юніті. Його незмінна популярність свідчить про здатність захоплювати гравців і надихати розробників розширювати межі можливого в інтерактивних розвагах. З розвитком технологій та зміною очікувань гравців одне залишається безсумнівним: жанр roguelike продовжуватиме розвиватися і

процвітати, забезпечуючи нескінченні пригоди для гравців, як старих, так і молодих.

Окрім розважальної цінності, рольові ігри також отримали визнання критиків за їхню здатність сприяти створенню нових сюжетів та змістовного ігрового досвіду. Поєднання процедурної генерації та випадкових подій створює сприятливий ґрунт для розгортання наративів, керованих гравцями, де кожне проходження гри стає унікальною історією виживання, дослідження та перемоги всупереч обставинам.

Підсумовуючи, можна сказати, що розквіт рольових ігор - це золотий вік творчості, інновацій та експериментів в ігровій індустрії. Завдяки особливому поєднанню процедурної генерації, механіки перманентної смерті та генерації випадкових подій, рогалики зайняли нішу, яка продовжує захоплювати гравців і надихати розробників розширювати межі можливого в інтерактивних розвагах.

## 1.2 Роль випадковості в ігрових застосунках

Випадкові події в геймдизайні - це непередбачувані події або ситуації в грі, які генеруються за допомогою алгоритмів або систем рандомізації. Ці події додають варіативності, виклику та захоплення ігровому процесу, вводячи несподівані елементи, на які гравці повинні реагувати та орієнтуватися.

Характер випадкових подій може сильно відрізнятися залежно від жанру, механіки та цілей гри. Випадкові події за своєю суттю є непередбачуваними, тобто гравці не можуть передбачити, коли і як вони відбудуться. Ця невизначеність тримає гравців у напрузі, оскільки вони повинні бути готовими адаптуватися до нових викликів або можливостей, що можуть з’явитись в будь-який момент.

Також випадкові події можуть охоплювати широкий спектр сценаріїв, від незначних змін у навколишньому середовищі до великих сюжетних поворотів. Вони можуть включати такі речі, як раптові зміни погоди, несподівані зустрічі з ворогом, знахідки ресурсів або точки розгалуження сюжету. Різноманітність випадкових подій гарантує, що ігровий процес залишається динамічним і свіжим

протягом декількох проходжень.

Випадкові події часто містять елементи ризику та винагороди, де гравці повинні зважувати свої варіанти і приймати рішення, виходячи з потенційних результатів. Деякі події можуть пропонувати цінні винагороди або переваги, якщо їх успішно пройти, тоді як інші можуть представляти значні проблеми або невдачі. Цей елемент ризику-винагороди додає ігровому процесу глибини та стратегічної складової.

І хоча випадкові події, за визначенням, знаходяться поза контролем гравця, вони все ж можуть надавати можливості для ігрової активності та осмисленого вибору. Гравці можуть мати можливість впливати на результат певних подій своїми діями, рішеннями або атрибутами персонажа, що дозволяє їм здійснювати певний контроль над непередбачуваними елементами ігрового світу.

Окрім цього, випадкові події можуть призвести до емерджентного ігрового досвіду, коли несподівані взаємодії між ігровими системами та механіками створюють унікальні та незабутні моменти. Ці емерджентні моменти сприяють поглибленню та реіграбельності ігор, оскільки гравці відкривають для себе нові стратегії, тактики та історії з кожним проходженням.

Хоча випадковість додає азарту та різноманітності ігровому процесу, для геймдизайнерів важливо збалансувати випадковість зі справедливістю. Випадкові події повинні покращувати загальний ігровий досвід, але при цьому не бути надмірно жорстокими чи несправедливими до гравців. Розробники можуть впроваджувати системи для пом’якшення впливу особливо негативних подій або гарантувати, що випадковість буде збалансована вмілою грою.

Загалом, випадкові події відіграють вирішальну роль у геймдизайні, додаючи невизначеності, виклику та нових ігрових можливостей. За умови ефективної реалізації вони підвищують залученість гравців, реграбельність та занурення, сприяючи загальному задоволенню від гри та її довготривалості.

Випадковість відіграє важливу роль у формуванні ігрового процесу та впливає на досвід гравців у широкому спектрі відеоігор. Чи то через процедурно згенеровані рівні, випадкові зустрічі або непередбачувані події, випадковість

вносить варіативність і невизначеність, які можуть глибоко впливати на те, як гравці взаємодіють з грою та сприймають її.

Коли гравці не можуть передбачити кожен результат або передбачити кожен виклик, вони змушені залишатися пильними та адаптивними, зберігаючи досвід свіжим та захоплюючим. Ця непередбачуваність сприяє відчуттю відкриття та дослідження, коли гравці переміщуються через динамічно створене середовище або стикаються з несподіваними перешкодами та можливостями. Більше того, випадковість вносить шар ризику та винагороди, що додає глибини та складності у процес прийняття рішень гравцями.

Гравці повинні зважувати потенційні переваги та недоліки своїх дій у світлі невизначених результатів, що призводить до стратегічного вибору та прорахованих ризиків. Наприклад, у рольовій грі гравці можуть вирішити дослідити незвідану територію в пошуках цінної здобичі, знаючи, що на цьому шляху вони можуть зіткнутися з могутніми ворогами або пастками.

Цей елемент ризику та винагороди додає напруження та азарту в ігровий процес, оскільки гравці повинні збалансувати своє прагнення до прогресу та винагороди з потенційними наслідками невдачі.

Як було описано вище, випадковість також сприяє виникненню нових вражень від гри, які визначають багато сучасних відеоігор. Коли випадкові елементи взаємодіють один з одним або з діями гравців у несподіваний спосіб, вони можуть створювати унікальні моменти, що запам’ятовуються, які відчуваються органічно і не за сценарієм. Ці несподівані моменти заохочують до творчості та експериментів, оскільки гравці відкривають для себе нові стратегії, тактики та рішення проблем шляхом спроб і помилок. Наприклад, у грі-пісочниці з процедурно згенерованими світами гравці можуть натрапити на приховані скарби або зустріти малоймовірних союзників, що призводить до виникнення наративів та історій, керованих гравцями. Однак випадковість в ігровому процесі не позбавлена викликів і пасток. Надмірна випадковість може призвести до розчарування і незадоволення, якщо гравці відчують, що їхній успіх чи поразка визначаються більше везінням, ніж навичками. Це може призвести до відчуття

несправедливості або свавілля, підриваючи відчуття майстерності та досягнення, якого багато гравців прагнуть в іграх. Крім того, випадковість може призвести до порушення балансу, якщо певні випадкові елементи непропорційно надають перевагу певним гравцям або стилям гри, або ставлять їх у несприятливе становище. Щоб пом'якшити ці проблеми, розробники ігор повинні ретельно збалансувати випадковість з іншими принципами дизайну, такими як ігровий процес, заснований на навичках, системи прогресу та агенції гравців. Впроваджуючи системи, які гарантують, що випадковість покращує, а не погіршує загальний досвід гравця, дизайнери можуть створювати ігри, які одночасно є складними і корисними, сприяючи залученню і задоволенню для гравців усіх рівнів кваліфікації.

Зрештою, вплив випадковості на ігровий процес і досвід гравця залежить від того, наскільки ефективно вона інтегрована в ширший дизайн гри і наскільки вона покращує основні механіки, теми і цілі гри.

Реалізація генерації випадкових подій в іграх передбачає використання різноманітних технік та алгоритмів, спрямованих на створення динамічного та непередбачуваного досвіду для гравців. Ці методи можуть варіюватися залежно від конкретних вимог гри та бажаного рівня випадковості. Ось кілька ключових підходів, які зазвичай використовують розробники ігор:

Генерація псевдовипадкових чисел (ГВЧ): Алгоритми ГПВЧ є фундаментальними для генерування випадкових подій в іграх. Ці алгоритми використовують математичну формулу для створення послідовностей чисел, які здаються випадковими, але насправді є детермінованими. Задавши початкове значення на вході, розробники можуть гарантувати, що послідовність випадкових чисел буде відтворюваною, що корисно для налагодження та тестування. Алгоритми ГПВЧ є універсальними і широко використовуються в розробці ігор завдяки своїй ефективності та передбачуваності.

Рандомізація на основі сіду: у багатьох іграх розробники використовують рандомізацію на основі сіду, щоб забезпечити послідовну випадковість у різних проходженнях або екземплярах гри. Генеруючи випадкове початкове значення на

початку кожної ігрової сесії або рівня, розробники можуть створювати унікальні варіації випадкових подій, зберігаючи при цьому певну послідовність і повторюваність. Цей підхід особливо поширений у процедурних системах генерації, де однакове початкове значення може створювати ідентичні ігрові світи або рівні.

Зважена рандомізація: зважена рандомізація дозволяє розробникам контролювати ймовірність певних подій, що відбуваються в грі. Призначаючи ймовірності або ваги різним результатам, розробники можуть впливати на частоту і розподіл випадкових подій, щоб досягти бажаних ігрових ефектів. Наприклад, у рольовій грі розробники можуть використовувати зважену рандомізацію, щоб визначити шанси натрапити на рідкісні предмети або ворогів залежно від таких факторів, як рівень гравця або місцезнаходження.

Системи запуску подій: у деяких іграх випадкові події запускаються на основі певних умов або тригерів у ігровому світі. Розробники можуть використовувати системи запуску подій для визначення критеріїв, коли мають відбуватися випадкові події, наприклад, дії гравця, умови навколишнього середовища або тригери, засновані на часі. Ретельно продумуючи ці системи запуску, розробники можуть створювати динамічний і чутливий ігровий процес, який занурює і захоплює.

Процедурна генерація: процедурні методи генерації зазвичай використовуються для створення випадкового контенту, такого як рівні, оточення або предмети в грі. Ці методи покладаються на алгоритми для створення контенту алгоритмічно на основі попередньо визначених правил і параметрів. Використовуючи процедурну генерацію, розробники можуть створювати величезні та різноманітні ігрові світи, які відчуваються унікальними та експансивними, надаючи гравцям безмежні можливості для досліджень та відкриттів.

Динамічне регулювання складності: випадкові події також можна використовувати для динамічного налаштування складності гри на основі результатів гравців або інших факторів. Вводячи випадкові елементи, складність

яких змінюється залежно від навичок гравця або його прогресу, розробники можуть створювати адаптивний ігровий досвід, який залишається складним і цікавим протягом всієї гри. Такий підхід дозволяє розробникам задовольнити потреби гравців з різними рівнями навичок та уподобаннями, гарантуючи, що гра залишається доступною та приємною для всіх гравців.

Таким чином, реалізація генерації випадкових подій в іграх передбачає поєднання технік і алгоритмів, призначених для створення динамічного, непередбачуваного та захопливого ігрового процесу. Використовуючи такі методи, як генерація псевдовипадкових чисел, рандомізація на основі сідів, зважена рандомізація, системи запуску подій, процедурна генерація та динамічне регулювання складності, розробники можуть створювати ігри, які будуть свіжими, цікавими та корисними з кожним проходженням.

### 1.3 Аналіз існуючих ігрових реалізацій

Хоча основні принципи перманентної смерті, процедурної генерації контенту та складного ігрового процесу залишаються основою жанру, сучасні розробники ігор постійно розширюють межі та експериментують з новими підходами. У цьому розділі ми розглянемо алгоритмічні інновації та потенційні недоліки деяких нещодавніх рогаликів:

1. Hades (Supergiant Games, 2020). На відміну від традиційних рольових ігор, де підземелля генеруються суто процедурно, в Hades використовується гібридний підхід. Основна схема підземного світу залишається відносно незмінною, але конкретні деталі, такі як зустрічі з ворогами, конфігурація кімнат і навіть варіанти діалогів, динамічно змінюються під впливом вибору гравця та його відносин з персонажами в рамках оповіді.

Алгоритмічно цю гру можна розбити на такі аспекти:

1) Мета-прогрес: гра відстежує прогрес гравця за допомогою різних валют і постійних поліпшень, подібних до дарів олімпійських богів. Вони переносяться між проходами, впливаючи на складність і потенційні нагороди, що зустрічаються в підземеллі.

2) Відносини між персонажами: оскільки гравець взаємодіє з NPC протягом проходження, його стосунки з цими персонажами розвиваються. Це може вплинути на типи доступних діалогів, потенційні побічні завдання і навіть на типи ворогів, що зустрічаються на основі сформованих прихильностей.

3) Динамічні події: розповідь розгортається через розмови та події, що відбуваються під час дослідження. Ці події можуть пропонувати цінні нагороди, розкривати фрагменти історії або навіть вводити нові виклики на основі поточного стану оповіді.

Рисунок 1.5 – Ігровий процес «Hades»

Сильні сторонами гри є персоналізація та стратегічна глибина. Такий підхід до процедурної генерації контенту, що ґрунтується на наративі, сприяє відчуттю персоналізації та інвестиції в історію. Кожна сесія відчувається як продовження всеохоплюючого сюжету, а вибір гравця впливає на розвиток подій.

Крім того, динамічна природа підземелля додає стратегічної глибини. Гравці повинні враховувати не тільки безпосередні бойові завдання, але й довгострокові наслідки своїх дій для сюжету. Вибір пріоритетності квестової лінії певного персонажа може розблокувати цінні знахідки або розкриття історії, але це також може призвести до зустрічі з більш складними ворогами, що належать до протилежних фракцій.

Окрім сильних сторін, також можна виділити такі недоліки, як баланс між

розповіддю та чистим ігровим досвідом

Хоча інтеграція наративу збагачує ігровий досвід багатьох гравців, вона може бути не всім до вподоби. Гравці, які віддають перевагу традиційній рольовій грі, що базується виключно на навичках, можуть вважати наративну складову нав'язливою або обмежуючою.

Крім того, залежність від заздалегідь визначеної сюжетної лінії може обмежити загальну процедурну варіативність у порівнянні з іграми з меншим впливом сюжету. Хоча структура підземелля динамічно змінюється, основна структура і типи ворогів залишаються дещо незмінними.

Так чи інакше, Hades демонструє захоплюючий потенціал сюжетної процедурної генерації в жанрі roguelike. Подальші дослідження та розробки можуть вдосконалити цей підхід.

Наприклад, впровадження більш динамічних змін до самої структури підземелля на основі наративного вибору може ще більше посилити взаємозв'язок між історією та ігровим процесом.

Вивчення потенціалу дійсно розгалужених гілок сюжету з кількома кінцівками, що залежать від вибору гравця, може запропонувати ще більшу реіграбельність і відчуття активності в історії.

2. Slay the Spire (MegaCrit, 2017). Slay the Spire, розроблена компанією MegaCrit, вирізняється тим, що вона є рогаликом, який вводить механіку побудови колоди в основний цикл гри. Цей інноваційний підхід вводить рівень прийняття стратегічних рішень та оцінки ризиків і винагород, захоплюючи гравців потенціалом для створення потужних та унікальних карткових синергій.

Система процедурної генерації в Slay the Spire фокусується на динамічному створенні пулу карт, з якого гравці можуть формувати свою колоду протягом партії. На цей пул впливають такі фактори, як обраний клас персонажа та конкретний шлях, пройдений через шпиль. Розбивка ключових механік виглядає приблизно так:

1) Пул карт: кожен клас персонажів має унікальний пул карт, що представляє їхні наступальні, оборонні та спеціальні можливості. Ці карти

пропонують різноманітні ефекти, від прямої шкоди до маніпулювання приростом енергії та управління скидом.

2) Відкриття карт: під час підйому по вежі, гравці мають можливість додавати карти до колоди. Це може статися завдяки перемозі над ворогами, купівлі карт у торговців або отриманню їх як винагороду за завершення сутичок.

3) Синергія та стратегія: основний ігровий процес обертається навколо побудови збалансованої колоди шляхом визначення синергії між різними картами. Гравці повинні враховувати такі фактори, як зниження вартості, комбо-ефекти та масштабування потенціалу шкоди, щоб створити колоду, здатну подолати все більш складних ворогів, що зустрічаються на верхніх поверхах.


Рисунок 1.6 – Ігровий процес «Slay the Spire»

Сильними сторонами Slay the Spire є стратегічна глибина та свобода дій гравця.

Елемент побудови колоди в Slay the Spire розвиває відчуття стратегічної глибини. Гравці не просто реагують на ситуації, а активно створюють двигун, який керує їхнім успіхом. Кожне рішення додати або прибрати карту з колоди має довгостроковий вплив на їхню здатність вирішувати майбутні проблеми.

Крім того, система побудови колоди надає гравцям високий ступінь свободи дій. Можливість керувати власною колодою виховує почуття власності та майстерності над можливостями свого персонажа. Експерименти з різними

комбінаціями карт і відкриття ефективних синергій можуть бути неймовірно корисними.

Як недоліки можна виділити так зване прокляття рандомної генерації чисел та відхід від звичних механік. Хоча механіка побудови колоди збагачує ігровий процес для багатьох, вона також вносить елемент випадковості, який може розчаровувати. Пул карт, пропонованих під час раунду, може сильно залежати від випадковості. Гравці можуть не натрапити на певні карти, які їм потрібні для створення ідеальної колоди, що потенційно може призвести до відчуття безпорадності та перешкоджати їхньому прогресу.

Крім того, зосередженість на створенні колоди може сподобатися не всім любителям жанру roguelike. Ті, хто віддає перевагу більш інтерактивному бойовому досвіду із заздалегідь визначеними здібностями персонажів, можуть вважити аспект побудови колоди громіздким і відволікаючим від основного ігрового циклу.

Так чи інакше, інноваційна механіка побудови колод у Slay the Spire прокладає шлях для захоплюючого розвитку жанру рогаликів. Можна виділити кілька потенційних напрямків для майбутніх досліджень:

Пом’якшення ГВЧ: Впровадження систем, що дозволяють гравцям маніпулювати пулом карт або гарантувати доступ до певних типів карт, може зменшити розчарування, пов’язане з покладанням на випадок.

Розширені можливості кастомізації колод: Більший контроль над розміром колоди або можливість повністю видаляти непотрібні карти може дати гравцям більше можливостей для вдосконалення своїх стратегій формування колод.

3. Nuclear Throne (Vlambeer, 2013). Nuclear Throne, розроблена компанією Vlambeer, використовує унікальний підхід до жанру roguelike, реалізуючи мінімалістичну естетику і ставлячи на перше місце складний ігровий процес.


Рисунок 1.7 – Ігровий процес «Nuclear Throne»

Система процедурної генерації в Nuclear Throne надає перевагу створенню структур, які сприяють швидкому розвитку подій та інтенсивним сутичкам з ворогом. На відміну від деяких рогаликів з величезними і розлогими підземеллями, Nuclear Throne обирає менші, більш тісно пов'язані рівні, які заохочують постійний рух і стратегічне використання зброї. Гру можна розбити на наступні ключові механіки:

1) Компактні рівні: процедурно згенеровані рівні відносно невеликі і заохочують до дослідження в обмеженому просторі. Таке стиснене середовище сприяє частим зустрічам з ворогом і постійному відчуттю терміновості.

2) Обмежена різноманітність ворогів: хоча в грі представлені різні типи ворогів, загальний пул ворогів залишається відносно обмеженим у порівнянні з деякими іншими рогаликами. Це дозволяє зосередитися на освоєнні основних бойових механік, а не на запам'ятовуванні конкретних схем атак ворога.

3) Зосередженість на зброї: різноманітний арсенал зброї з унікальними механіками є ключовим аспектом гри. Гравці можуть знаходити і використовувати різну зброю протягом проходження, кожна з якої вимагає різного підходу до бою.

Сильними сторонами гри є ігровий процес та майстерність, що ґрунтуються виключно на навичках гравця.

Мінімалістичний дизайн Nuclear Throne та зосередженість на основних механіках перерахованих вище, формують відчуття чистого ігрового процесу, заснованого на навичках. Гравці покладаються на свої рефлекси, тактичне позиціонування та майстерність поводження зі зброєю, щоб просуватися вперед. Менше відволікаючих чинників та складних механізмів, які потрібно враховувати, що дозволяє гравцям по-справжньому відточувати свої навички та відчувати задоволення від подолання все більш складних ворожих хвиль.

Крім того, зосередженість на майстерності розвиває почуття досягнення. Успішне проходження забігу та перемога над фінальним босом вимагає глибокого розуміння механіки гри та вміння адаптуватися до швидкозмінних ситуацій.

Головними недоліками можна виділити доступність, реіграбельність та стиль.

Висока крива складності Nuclear Throne може бути палицею з двома кінцями. Хоча вона пропонує цікавий досвід для досвідчених гравців, вона може відштовхнути новачків у жанрі. Невблаганна перманентна смерть та відсутність шкали складності можуть створити круту криву покращення навичок, яка не дає деяким гравцям продовжувати гру далі і змушує зупинятись на більш легких етапах гри.

Крім того, обмежене розмаїття ворогів і залежність від меншого набору заздалегідь визначених елементів може знизити реіграбельність порівняно з рогаликами з більш розвиненими системами процедурної генерації. Хоча фокус на майстерності пропонує велику кількість викликів для дослідження, гравцям, які шукають світ з новими ворогами, що постійно розвивається, Nuclear Throne може здатись дещо повторюваним після тривалої гри.

Також мінімалістичний візуальний стиль може відштовхнути сучасних геймерів відсутністю деталізованого та наповненого дизайну, що часто не витримує постійні перегони з конкурентами в інших стилях.

Але в будь-якому випадку Nuclear Throne є доказом потенціалу мінімалістичного дизайну та високої складності в жанрі рольової гри. Потенційні

напрямки для розвитку можуть виглядати приблизно так:

Режими змінної складності: впровадження налаштувань складності або механік, які дозволять гравцям поступово підвищувати рівень складності, може зробити гру більш доступною для ширшої аудиторії.

Розширене розмаїття ворогів або процедурно згенеровані риси ворогів: додавання нових типів ворогів або процедурне генерування варіацій існуючих типів ворогів може підвищити реіграбельність, вводячи нові виклики, не відхиляючись при цьому від основного фокусу на ігровому процесі, заснованому на навичках гравця.

Усунувши ці потенційні недоліки та дослідивши способи розширення основних механік, розробники можуть створити ще більш захопливий та доступний досвід гри на високому рівні складності, який задовольнить ширший спектр уподобань гравців.

## 1.4 Сучасний стан ігрової індустрії

У 2024 році жанр roguelike продовжує процвітати. Однак геймплей постійно змінюється, оскільки розробники постійно знаходять шляхи інтеграції інноваційних механіки. У цьому розділі розглянуто поточний стан розвитку індустрії, ключові тенденції та приклади ігор, які ілюструють ці досягнення.

Тенденція 1: Інтеграція наративу та осмислений вибір. Часи статичних сюжетів або мінімальної оповіді в рогаликах минули. Наративні рогалики використовують алгоритми процедурної генерації контенту, які включають елементи сюжету. Це означає, що підземелля, сутички і навіть типи ворогів більше не є статилими, а можуть змінюватися під впливом рішень гравця та його стосунків з персонажами в рамках сюжету. Можна уявити, як, наприклад, дослідження підземелля в Hades приводить до кімнати, тематично пов’язаної з певним божеством, на основі того, як гравець взаємодіяв з цим богом у попередніх проходженнях. Такий динамічний підхід підвищує рівень дослідницького досвіду, створюючи відчуття несподіванки та зв’язку з оповіддю.

Наративні рольові ігри часто роблять сильний акцент на взаємовідносинах

персонажів. Гравці будують зв'язки з неігровими персонажами через вибір діалогів, дії під час дослідження і навіть бойові рішення. Ці стосунки розвиваються протягом декількох проходжень гри, значно впливаючи на сюжетну лінію. Уявіть, що ви подружилися з торговцем в одному з проходжень гри, а потім зустрілися з ним пізніше, отримавши унікальний квест або знижку, засновану на ваших минулих взаємодіях. Це додає ваги рішенням гравця, виховуючи почуття наслідків та інвестиції у світ.

Одним з найбільш захоплюючих аспектів цього тренду є концепція вибору та його наслідків. Рішення гравця мають відчутний вплив на розвиток сюжету. Вибір, зроблений під час досліджень, бойових зіткнень чи навіть, здавалося б, буденних діалогів, може відкрити нові сюжетні лінії, вплинути на прихильність персонажів і, зрештою, визначити долю світу чи головного героя. Наприклад, у такій грі, як Pyre (Supergiant Games, 2017), вибір персонажів, з якими ви товаришуєте, та рішення, які ви приймаєте під час випробувань, можуть призвести до дуже різних фіналів, заохочуючи до багаторазового проходження гри, щоб відчути повне занурення в сюжет.

Інтеграція наративу в ігровий цикл може наповнити традиційну механіку перманентної смерті додатковим змістом. Навіть зазнавши поразки, гравці можуть відчути прогрес у розвитку стосунків, відкритті фрагментів історії та залишенні свого сліду у світі. Це пом'якшує розчарування, пов'язане з смертю в грі, пропонуючи відчуття сюжетного завершення навіть перед обличчям поразки.

Мабуть, найбільш значущою перевагою інтеграції наративу є створення більш глибокого емоційного зв'язку між гравцями та ігровим світом. Вплітаючи елементи історії в ігровий процес, розробники створюють сюжети і персонажів, в які гравці вкладають душу. Долі персонажів і світу можуть викликати справжній емоційний резонанс, створюючи більш захоплюючий і незабутній досвід.

Хоча наративна інтеграція пропонує захоплюючі можливості, деякі виклики залишаються. Так вкрай важливо підтримувати баланс між захопливою розповіддю та складним, процедурно згенерованим ігровим процесом. Надмірна

увага до оповіді не повинна затьмарювати основні принципи жанру. Гравці все ще очікують на задоволення від циклу досліджень, боїв та прийняття стратегічних рішень.

Тенденція 2: Злиття жанрів. Жанр переживає хвилю жанрового злиття. Slay the Spire ілюструє цю тенденцію, впроваджуючи механіку колодобудування у формулу рольової гри. Гравці збирають колоду карт протягом свого ходу, стратегічно вибираючи карти, щоб визначити свої наступальні та оборонні можливості. Таке поєднання додає рівень стратегічного прийняття рішень та оцінки ризиків і винагород, що приваблює ширшу аудиторію гравців, які полюбляють стратегічні карткові ігри.

Тенденція 3: Алгоритмічний прогрес. Розробники постійно вдосконалюють алгоритми процедурної генерації, щоб створювати більш динамічні та захопливі середовища. Наприклад, такі ігри, як Returnal (Housemarque, 2021), використовують елементи емерджентного геймплею, створюючи синергію між небезпеками навколишнього середовища та поведінкою ворога. Такий підхід гарантує, що кожне проходження гри ставить перед гравцем унікальні виклики і вимагає від нього адаптації.

Тенденція 4: Доступність та зменшення розчарування. Усвідомлюючи потенційний бар'єр входу для новачків, деякі розробники впроваджують функції, які роблять рогалики більш доступними. Hades пропонує поблажливу механіку з постійними оновленнями, які переносяться між проходженнями, забезпечуючи відчуття довготривалого прогресу навіть в умовах перманентної смерті. Крім того, деякі представники жанру досліджують змінні налаштування складності або впроваджують механіки, які дозволяють гравцям поступово підвищувати рівень складності, полегшуючи криву навчання для новачків.

Хоча ці тенденції демонструють життєздатність жанру, деякі проблеми залишаються. Досягнення балансу між впровадженням свіжої механіки та збереженням основних принципів roguelike має вирішальне значення. Надмірна залежність від процедурної генерації може призвести до браку ручної роботи над контентом, що потенційно призводить до відсутності повноцінного ігрового

досвіду. Крім того, ключовим моментом залишається збалансування складності та пом’якшення розчарування.

Отже, жанр рольових ігор у 2024 році – це яскравий майданчик для інновацій та експериментів. Розробники розширюють межі, інтегрують наративні елементи, змішують жанри та вдосконалюють алгоритми процедурної генерації. Усуваючи потенційні недоліки та досліджуючи нові напрямки, жанр roguelike може продовжувати захоплювати гравців та дарувати постійно розвиваючий досвід на довгі роки.

# РОЗДІЛ 2.
# КЛЮЧОВІ АСПЕКТИ РОЗРОБКИ ІГРОВОГО ЗАСТОСУНКУ

## 2.1 Основні механіки для рогаликів

У рогаликах з процедурно згенерованими рівнями динамічна взаємодія механік сприяє захопливому та цікавому ігровому досвіду.

Дослідження стає життєво важливим компонентом, завдяки процедурній генерації рівнів. Кожне проходження представляє свіже середовище з унікальним плануванням, кімнатами та завданнями. На всіх рівнях розкидана система поліпшення, яка слугує рушієм для розвитку персонажа. Зброя та обладунки з різною рідкістю та властивостями поступово відкриваються, що дозволяє гравцям налаштовувати наступальні та оборонні можливості свого персонажа. Крім того, витратні предмети, наприклад такі, як зілля та сувої, дають тимчасові бафи, стратегічні переваги або екстрене зцілення, що ще більше розширює тактичні можливості гравця.

Бойові дії, як покрокові, так і в реальному часі, ґрунтуються на основному наборі механік, зосереджених на управлінні ресурсами та виконанні дій. Очки здоров’я відображають життєву силу персонажа, яка виснажується з кожною отриманою атакою. Ресурси, такі як мана або витривалість, визначають частоту та ефективність спеціальних здібностей або дій персонажа. Вміле управління цими ресурсами набуває першочергового значення, оскільки їх виснаження може зробити гравців вразливими або позбавити їх можливості використовувати основні навички.

Введення випадкових подій вносить елемент непередбачуваності, постійно тримаючи гравців у напрузі. Ці події, спричинені діями гравця, досягненням певних етапів рівня або чистою випадковістю, можуть варіюватися від випадкових зустрічей з прихованими торговцями, що пропонують потужне спорядження, до небезпек навколишнього середовища, які змушують тактично відступати або змінювати планування ігрової зони. Цей елемент несподіванки вимагає динамічної адаптації гравців, вимагаючи від них переоцінки

запланованих стратегій і прийняття рішень за лічені секунди, виходячи з нових обставин, що з’являються в результаті події.

Нарешті, основна механіка перманетної смерті додає значної ваги кожному рішенню, прийнятому гравцем. На відміну від традиційних ігор з постійним прогресом, у roguelike одна-єдина помилка, наприклад, невчасна сутичка або невміння орієнтуватися в пастці, може призвести до загибелі персонажа і змусити почати все з самого початку. Ця постійна загроза втрати прогресу спонукає гравців опановувати методи дослідження, оптимізувати використання ресурсів і приймати стратегічні бойові рішення, щоб подолати виклики, пов’язані з процедурно згенерованим світом і мінливістю випадкових подій.

### 2.2 Опис ігрової механіки та архітектури застосунку

Сюжет гри досить тривіальний, але зрозумілий. Людина опиняється в дивному сні, і намагається прокинутись, а для цього їй потрібно відкривати двері, долати свої страхи, використовувати логіку і вірити в свою вдачу.

Основний геймплей гри уявляє собою досить легку суміш рогалика та ігор жанру «point-and-click». Гравець опиняється в незрозумілій кімнаті і все, що він може робити - це просто оглядатись навколо і клацати мишкою в будь-яку область. Оглянувши кімнату, можна буде помітити, що перед ним буде декілька дверей різного матеріалу, стану та, в цілому дизайну. Клацнувши на одну з них, гравець переходитиме до наступної кімнати, де, в залежності від вибору двері, відбудеться або не відбудеться випадкова подія. Основною ціллю гри є вибратись з цього лабіринту.

Для кращого розуміння, що саме буде в застосунку, варто розписати які механіки потрібно в ньому реалізувати.

Основні механіки:

1) Перманентна смерть. Як і в усіх рогаликах, однією з важливих для цього жанру механік, буде реалізація перманентної смерті, без можливості збереження прогресу.

2) Процедурна генерація рівнів. Рівні генеруються випадковим чином

під час кожного проходження, щоразу пропонуючи новий досвід і заохочуючи до дослідження. В подальшому детальніше розберемо якими алгоритмами можна користуватись для реалізації цієї механіки.

Схема виглядає наступним чином: визначення початкових параметрів генерації на кшталт розміру рівня, кімнати з подіями, з пастками і т.д. – створення кімнат різного типу – створення графової структури, де кожна кімната є вузлом а двері між ними є ребрами – розміщення виходів в кімнатах так, щоб забезпечити можливість переходу між ними – додавання випадкових подій – знаходження шляху від входу до виходу, який можна будувати за допомогою різних алгоритмів(наприклад алгоритм Дейкстри) – завершення генерації.

3) Лічильник часу. Зверху на екрані гравця буде знаходиться лічильник, який показує скільки часу займає його проходженння.

4) Шкала глузду. У грі реалізовано ресурс – глузд. В залежності від того, на якому рівні знаходиться показник шкали, гравець може отримувати різні бафи та дебафи, починаючи від цільного яскравого зображення і закінчуючи смертю за певних обставин.

5) Система випадкових подій:

- Пул подій. Набір заздалегідь визначених подій з різним впливом на ігровий процес(наприклад двері з кодом, монстри за дверима, падіння за дверима, музичні кімнати та ін.)

Фактично, пул подій являє собою скінченну колекцію попередньо визначених подій з відповідними ефектами на ігровий процес. Ці події можна класифікувати на основі їхнього впливу на гравця:

- Корисні події: забезпечують тимчасові позитивні ефекти, такі як, наприклад: підвищення характеристик гравця, зменшення кількості предметів або регенерація здоров'я.

- Події випробування: створюють труднощі, такі як, наприклад, зустрічі з ворогами, пастки або небезпеки навколишнього середовища.

- Динамічні події: змінюють динаміку рівня за допомогою таких механізмів, як часові обмеження, зміни навколишнього середовища або змінена

фізика.

- Механіка тригерів. Події можуть бути викликані діями гравця (наприклад, гравець довгий час не обирає двері і в нього починається особлива подія), проходженням рівня (наприклад, коли гравець буде досягати кожної п’ятнадцятої кімнати, його буде очікувати визначена зона, яка повторюється з проходження до проходження) або випадковим збігом обставин з регульованою ймовірністю.

Механіка тригерів визначає умови, за яких активується подія з пулу подій.Відбувається це наступним чином:

Розглянемо приклад, коли подія запускається певною дією гравця.

Наприклад, гравець вирішує взагалі не обирати двері і, коли лічильник часу визначає, що гравець перевищив певний часовий поріг без вибору дверей, то може викликати спеціальну подію, яка змусить гравця все ж обрати двері, або виконати іншу дію, яка матиме свої наслідки.

Або ж події активуються після завершення рівня. Це може включати в себе створення певної зони подій після досягнення певної віхи рівня (наприклад, кожна 15-та кімната, як було описано раніше).

Також є варіант, коли події запускаються із заздалегідь визначеною ймовірністю. Це дозволяє відбуватися подіям динамічно, не покладаючись виключно на дії гравця або проходження рівня, що сприяє покращенню ігрового досвіду за рахунок неочікуваності.

- Вплив події. Події можуть надавати переваги (тимчасові ефекти або різної якості предмети), створювати труднощі (вороги, пастки та перешкоди) або змінювати динаміку рівня (наприклад, події, коли в тебе є лише 5 секунд на вибір двері).

Цей вплив можна класифікувати залежно від типу події:

Наприклад, ми можемо визначити величину користі від корисної події. Це може бути кількісно визначено як тривалість підвищення характеристик гравця, рідкість випадання предметів або кількість відновленого здоров'я.

Так само ми можемо визначити, наскільки важкими будуть випробування,

які з’являтимуться перед гравцем. Дані випробування можуть вимірюватися кількістю та силою ворогів, що з'являються, розміром і складністю пасток або серйозністю небезпек навколишнього середовища.

Також можна впливати на зміни динаміки рівня, спричиненої динамічною подією. Це може бути кількісно визначено як встановлений часовий ліміт, ступінь змін у навколишньому середовищі або величина зміненої фізики.

- Масштабування впливу. Вплив подій можна масштабувати на основі прогресу гравця або рівня персонажа, щоб підтримувати збалансовану складність.

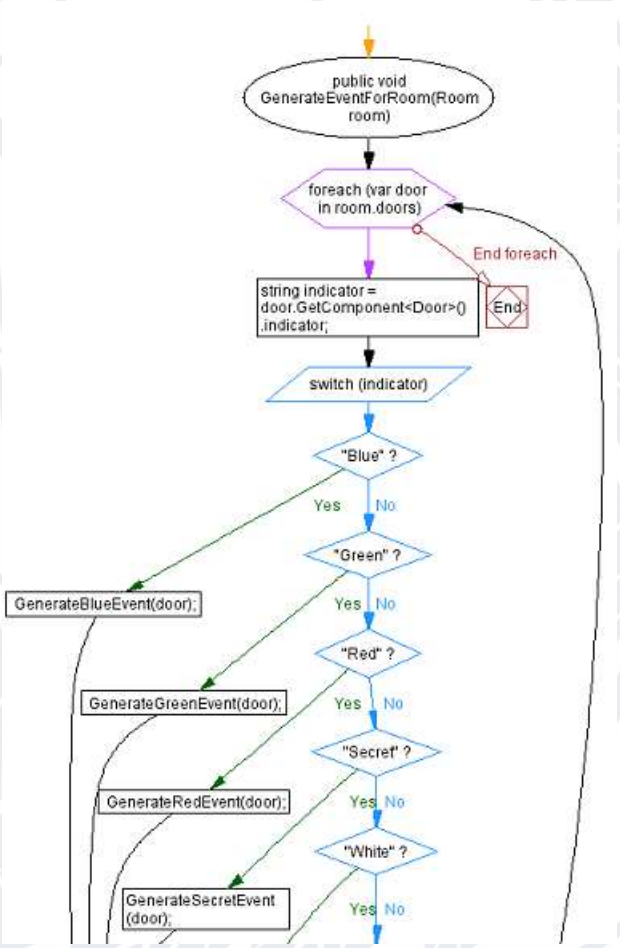


Рисунок 2.1 Блок-схема системи генерації випадкових подій

Також варто враховувати, що випадкові події повинні тематично відповідати загальному світу та історії рогалику, щоб забезпечити цілісний досвід. Окрім цього, вони повинні пропонувати вибір, який матиме значення

(наприклад, ризик проти винагороди), щоб підвищити активність та залученість гравців. Ну і звісно пул подій та механіки їх запуску повинні бути достатньо різноманітними, щоб забезпечити релевантність досвіду під час проходження гри.

### 2.3 Аналіз середовища розробки

Головним середовищем розробки для роботи є Unity, тому далі детальніше опишу дане середовище та технології. Найбільшим конкурентом, який існує на ринку ігрових рушіїв є Unreal Engine[16], який також має велику кількість можливостей для досліджень та створення застосунків, втім для реалізації даного проєкту, було вирішено, що доцільніше використовувати Unity.

Означимо переваги і можливості Unity.

C# - основна мова для програмування ігрових механік. Система сценаріїв Unity дозволяє визначати рух персонажів, бойові системи, взаємодію предметів тощо. [17-19]

В Unity можна створювати процедурно згенеровані підземелля за допомогою інструментів керування сценами. Заготовки, які є попередньо сконфігурованими ігровими об’єктами, стають основою для формування рівнів. За допомогою скриптів можна динамічно генерувати префаби, створюючи новий макет кожного разу під час гри.

Для рольової гри на основі сітки ідеально підійде система 2D тайлмапів Unity. Вона дозволяє визначати типи плиток (підлога, стіни і т.д.) і дозволяє процедурно розфарбовувати їх для створення підземель. У 3D-середовищі система Terrain пропонує інструменти для створення ландшафтів, які можуть бути корисними для створення органічних підземель без сітки.

В рушії Unity можна реалізувати реалістичні рухи та взаємодії. Він визначає, як ваш персонаж стикається з навколишнім середовищем та ворогами.

В Unity є система анімацій. Можна імпортувати 2D-скрипти або 3D-моделі з попередньо визначеною анімацією, або створювати власні за допомогою інструментів анімації Unity. Характерною рисою рогаликів є їхній світ, що

постійно змінюється. Саме тут на допомогу приходить процедурна генерація рівнів та інструменти для її реалізації.

Наприклад, Perlin Noise або Simplex Noise, можна використовувати для генерації випадкових макетів підземель. Ці функції створюють «початкове» значення, яке диктує загальну структуру, забезпечуючи певний рівень узгодженості, пропонуючи при цьому достатню варіативність.

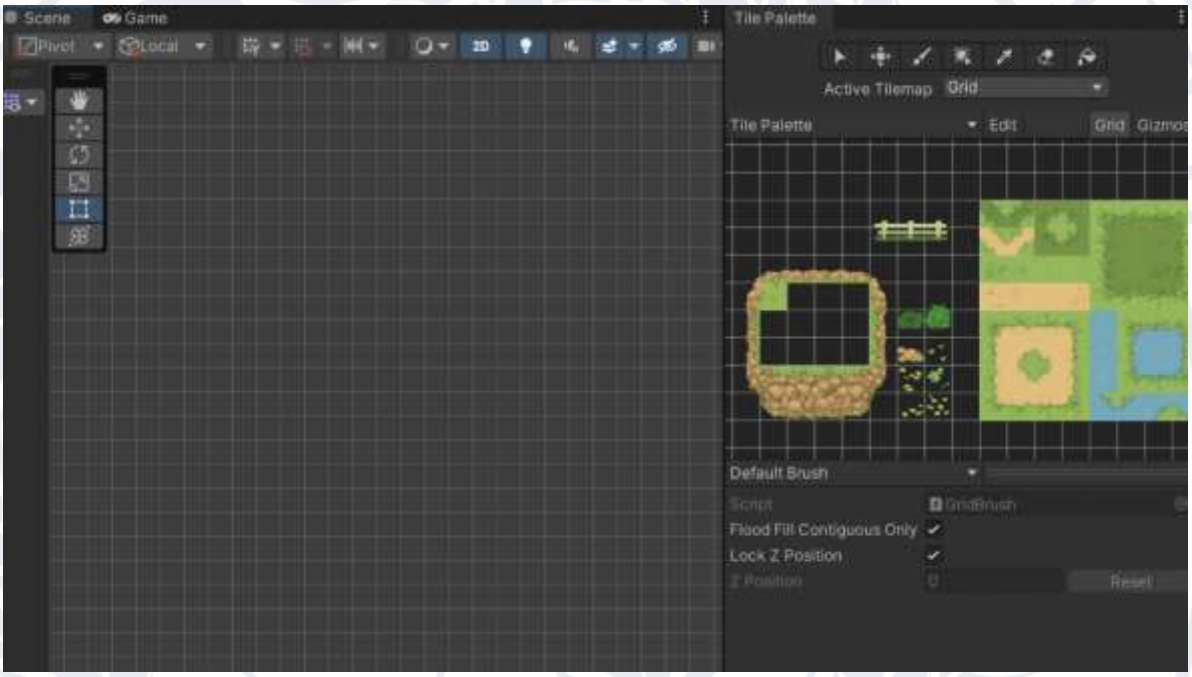


Рисунок 2.2 Система 2D тайлмапів Unity

Алгоритми на кшталт рекурсивного відстеження або клітинних автоматів можна використовувати для створення більш складних і цікавих макетів підземель. Ці алгоритми визначають правила з’єднання кімнат, розгалуження коридорів і розміщення спеціальних об’єктів, таких як магазини або кімнати босів.

Також в Unity можна використовувати пакети активів для керування величезною кількістю контенту (кімнат, ворогів, предметів), необхідного для процедурної генерації. Це дозволяє ефективно завантажувати та вивантажувати активи на основі згенерованого рівня, покращуючи продуктивність.[20]

І найголовніше, спільнота Unity є великою та дружньою. Це дає можливість

використовувати онлайн-форуми, навчальні посібники та магазини ресурсів, щоб знайти рішення, натхнення та готові ресурси, які можуть прискорити процес розробки.

Окрім Unity, важливим інструментом є й Adobe Photoshop, який є одним з найкращих застосунків у сфері цифрового дизайну, особливо впливовим у створенні візуальних ресурсів для відеоігор.

За своєю суттю Photoshop - це редактор растрової графіки. Растрова графіка представляє цифрові зображення у вигляді сітки пікселів, кожен з яких містить інформацію про колір і прозорість. Photoshop надає повний набір інструментів для маніпулювання цими пікселями, що дозволяє художникам створювати та редагувати широкий спектр візуальних елементів, які мають вирішальне значення для геймдизайну. У фотошопа є велика кількість переваг:

1. Шари: можна уявити стос прозорих аркушів, кожен з яких містить окремий візуальний елемент. Такий багаторівневий підхід у Photoshop дозволяє здійснювати паралельне редагування. Зміни, внесені до одного шару, не змінюють назавжди підлеглі шари, що забезпечує гнучкість і можливість переглядати творчі рішення.

2. Канали: кожен шар може складатися з декількох каналів. Ці канали зазвичай представляють інформацію про червоний, зелений, синій (RGB) кольори, але також можуть містити додаткові канали для таких факторів, як альфа (прозорість) або маски виділення. Канали забезпечують детальний контроль над візуальними властивостями зображення.[21]

3. Інструменти виділення: Photoshop пропонує потужний набір інструментів виділення, які дозволяють художникам ізолювати певні ділянки зображення для точкового редагування. Найпоширеніші інструменти виділення включають маркери (прямокутне або еліптичне виділення), ласо (виділення від руки) та інструмент «чарівна паличка» (автоматичне виділення на основі схожості кольорів).

4. Трансформації: виділеними областями можна маніпулювати за допомогою функцій масштабування, обертання, нахилу та деформації. Ці

трансформації дозволяють точно позиціонувати і формувати візуальні елементи в ігровому середовищі.

5. Фільтри та налаштування: велика бібліотека фільтрів та інструментів налаштування дозволяє художникам покращувати або змінювати візуальні характеристики зображення. До них відносяться фільтри для розмиття, підвищення різкості, зменшення шуму, корекції кольору та ефектів освітлення. Налаштування забезпечують неруйнівний контроль над такими факторами, як яскравість, контраст, відтінок і насиченість.

6. Пензлі: Photoshop пропонує величезну бібліотеку налаштовуваних пензлів, які імітують традиційні засоби, такі як олівці, фарби та аерографи. Ці пензлі дозволяють художникам створювати намальовані вручну текстури, концепт-арти та інші ілюстративні елементи для ігрових світів.

7. Текстури: текстурування пласких поверхонь вдихає життя в ігрове середовище. Photoshop дозволяє створювати, імпортувати та маніпулювати текстурами, які додають деталізації та глибини 3D-моделям у грі.

8. Ефекти шарів та режими змішування: ефекти шарів надають неруйнівні методи для додавання візуальних покращень, таких як тіні, зовнішнє світіння та внутрішні скоси. Режими накладання керують взаємодією нижчих шарів із поточним шаром, уможливлюючи композиційні техніки для створення реалістичних і візуально привабливих елементів.

9. Універсальність і потужність Photoshop роблять його безцінним інструментом для геймдизайнерів і художників з кількох причин:

10. Крос-платформна сумісність: активи Photoshop можна легко експортувати у різних форматах, сумісних з різними рушіями для розробки ігор, що сприяє безперебійному робочому процесу.

11. Паралельне редагування: багаторівневий підхід і можливості паралельного редагування дозволяють творчо досліджувати і легко вносити зміни без шкоди для основної роботи.

12. Широкі можливості кастомізації: велика бібліотека пензлів, фільтрів та ефектів дозволяє художникам розробляти унікальні візуальні стилі та втілювати

свої творчі бачення в ігровому світі.

13. Інтеграція з іншими інструментами для дизайну: Photoshop легко інтегрується з іншим програмним забезпеченням для дизайну, таким як Adobe Illustrator (векторна графіка), для спрощеного створення ресурсів або Adobe After Effects для створення анімацій.

# РОЗДІЛ 3.
# ДЕМОНСТРАЦІЯ ТА ОПИС ІГРОВОГО ЗАСТОСУНКУ

## 3.1 Алгоритми генерації випадкових подій

Система подій у рольовій грі використовує декілька основних алгоритмів для керування потоком ігрових подій:

1. Пошук в ширину або Breadth-First Search (неявний):

Система подій у рольових іграх, хоча на перший погляд здається простою, спирається на взаємодію алгоритмів, які організовують ось цей ефект активності, що розгортається в ігровому світі. Одним з таких алгоритмів, хоча і не реалізованим явно у вигляді традиційної структури даних, є Breadth-First Search (BFS). Ця концепція відіграє вирішальну роль у забезпеченні плавного та логічного потоку подій у грі.

Уявимо чергу подій як дерево, що постійно розвивається. Кожна подія, що спрацьовує під час ігрового процесу, стає вузлом цього дерева, а нові події, викликані вже існуючою подією, виступають як її «дочірні» гілки, що відгалужуються від батьківського вузла. Коли система подій обробляє події, вона ітераційно проходить через чергу у певному порядку. Цей порядок дуже нагадує обхід метафоричного дерева подій методом Breadth-First Search. BFS диктує, що всі вузли на певному рівні (відстані від початкової точки) повинні бути відвідані і оброблені, перш ніж переходити до вивчення вузлів на наступному рівні.

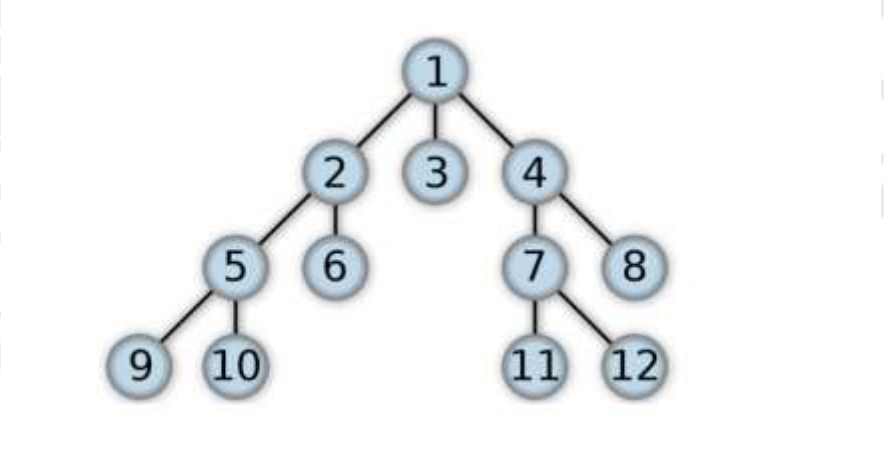


Рисунок 3.1 – Схема обробки ігрових подій BFS

На рисунку 3.1 зображено дерево чисел від 1 до 12, де кожне з них представлено вузлом дерева. Таким чином вузли утворюють ієрархічну стрктуру, де кожен вузол має один батьківський вузло і або не має, або має декілька дочірніх вузлів.

Фактично алгоритм працює так: проводиться ініціалізація, в ході якої всі наявні вузли позначаються, як невідвідані і обирається випадковий кореневий вузол. Наступним етапом є обхід дерева: відвідується кореневий вузол і для кожного дочірнього вузла визначається чи був він відвіданим: якщо так, то відбувається перехід до наступного вузла; якщо ні, то відвідується даний вузол. І фінальним етапом є представлення випадкової послідовності чисел, які позначають порядок, у якому вони були відвідані.

Наприклад, початковим вузлом є вузол 1(початкова кімната, з якої починається гра). Кожен вузол має дочірні вузли, які представляють собою кімнати і гравець має можливість потрапити в якусь із них. В кожному з вузлів може є певно ймовірність з появою однієї з кімнат з подією (до прикладу, 40%, що з’явиться кімната з головоломкою в вузлі 2, 8%, що з’явиться кімната з тінню або пасткою в вузлі 5 і вузол 8 може бути кімнтатою з відпочинком, яка з’являється кожну 8 кімнату)

Відповідно, обираючи двері, гравець обирає шлях до одного з дочірніх вузлів. Наприклад, в нього є можливість з вузла 1 перейти в вузли 2, 3 або 5, де будь-який з варіантів буде вести до якоїсь випадкової події. Одна з особливостей розробленого застосунку полягає в тому, що у гравця не буде можливості повністю обійти дерево. Фактично, він матиме змогу рухатись лише по невідвіданих вузлах вперед. Але повернемось до BFS.

Така обробка, подібна до BFS, має кілька переваг для системи подій. По-перше, вона гарантує завершення. Систематично проходячи через чергу, система гарантує, що кожна подія врешті-решт дочекається своєї черги, запобігаючи ситуаціям, коли деякі події постійно відсуваються назад новими, що додаються до черги. По-друге, BFS сприяє впорядкованому виконанню подій. Події обробляються в тому ж порядку, в якому вони були викликані, підтримуючи

логічний потік подій в ігровому світі. Це має сенс, якщо врахувати причинно-наслідковий характер ігрового процесу. Уявимо, що ворог атакує нас, викликаючи подію «Отримати ушкодження». BFS гарантує, що ця подія буде оброблена і наше здоров'я буде віднято, перш ніж потенційно запустити подію «Підвищення рівня», якщо вона також чекає в черзі - набагато логічніша послідовність порівняно з протилежною. Нарешті, BFS природно узгоджується з тим, як зазвичай функціонують ігрові цикли в roguelike. Кожна ітерація ігрового циклу обробляє події в черзі (поточний рівень у нашому метафоричному дереві), ефективно досліджуючи «дерево подій» рівень за рівнем у систематичний спосіб.

Однак важливо визнати, що BFS також має обмеження. Хоча вона гарантує, що всі події будуть врешті-решт оброблені, вона не розставляє пріоритети. Якщо критичні події, такі як смерть гравця, будуть поховані під купою менш термінових, таких як отримання предметів, це може призвести до затримок в обробці цих критичних ситуацій. Саме тут окремий алгоритм визначення пріоритетів, наприклад, черга пріоритетів, може стати цінним доповненням до системи подій. Крім того, важливо ретельно розробити логіку обробки подій, щоб уникнути ситуацій, коли події безперервно викликають одна одну, створюючи нескінченний цикл в обробці. Тайм-аути або запобіжники в обробниках подій можуть допомогти запобігти таким сценаріям.

На закінчення, хоча BFS не є традиційним алгоритмом, явно закодованим в системі подій, він забезпечує потужну концептуальну основу для розуміння того, як система справляється з постійно зростаючою чергою подій у грі, подібній на рольову гру. Забезпечуючи повне і логічне виконання подій, BFS відіграє життєво важливу роль у підтримці плавного і захоплюючого ігрового процесу для гравців.

2. Вказівник на функцію:

У рольових іграх, де світ реагує і розвивається на основі дій гравця, надійна система подій має важливе значення. Але як гра може ефективно визначити, яку дію слід виконати, коли відбувається певна подія? Саме тут у гру вступають покажчики функцій на основі делегатів, які виступають невидимим клеєм, що

зв’язує події з відповідними обробниками.

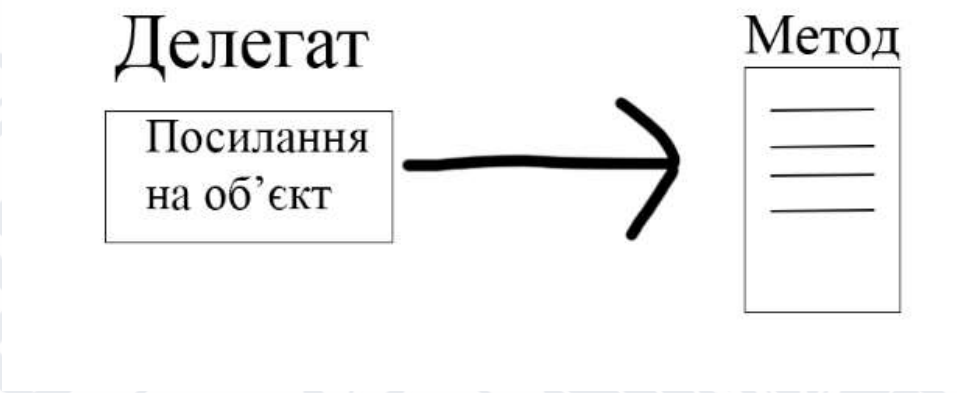


Рисунок 3.2 – Схема як працюють делегати

Уявимо жвавий ринок у якому-небудь рогалику. Наш персонаж наштовхується на торговця, що викликає подію «Взаємодія». Система подій, отримавши цю подію, повинна знати, що робити далі. Ось тут і з’являються покажчики функцій на основі делегатів.

У C# делегати виконують роль покажчиків функцій. Коли ми визначаємо тип події (наприклад, «Взаємодія»), ми також можемо визначити делегат, який вказує сигнатуру функції, що має обробляти цю подію. Ця сигнатура визначає параметри, які отримає функція-обробник (найчастіше сам об’єкт події), і тип повернення (найчастіше void). Тепер, коли спрацьовує подія «Взаємодія», система звертається до попередньо визначеної карти (часто реалізованої у вигляді словника), яка пов’язує типи подій з відповідними делегатами. Цей делегат, у свою чергу, вказує на конкретну функцію, відповідальну за обробку подій «Взаємодія». Потім система викликає цю функцію, передаючи об’єкт події як аргумент, ефективно запускаючи відповідний курс дій - можливо, ініціюючи діалог з продавцем.[22-28]

Підхід динамічного делегування пропонує декілька переваг для системи подій:

- Модульний дизайн: відокремлення типів подій від їх обробників робить

код більш модульним і легшим в обслуговуванні. Можна додавати нові типи подій та їх обробники, не змінюючи існуючий код. До прикладу, можна додати нову подію «Спроба кишенькової крадіжки». Просто визначається новий тип події, для неї створюється функція-обробник і далі вона додається до карти делегатів. Система подій автоматично розпізнає і безперешкодно обробляє цю нову подію.

- Гнучкість: покажчики функцій на основі делегатів дозволяють гнучко обробляти події. Можна підписувати декілька функцій на один і той самий тип події. Це уможливлює більш складні сценарії, де різні частини гри можуть бути зацікавлені в одній і тій самій події, але повинні виконувати різні дії. Наприклад, подія «Атака» може викликати обробник для нанесення шкоди цілі, тоді як інший обробник може перевірити смерть і викликати подію «Кінець гри», якщо це необхідно.
- Роз’єднання: цей підхід сприяє слабкому зв’язку між різними частинами ігрового коду. Обробникам подій не потрібно знати нічого конкретного про об’єкт, який викликав подію, або про загальний стан гри. Вони просто оперують даними, наданими об’єктом події, що робить код більш зручним для супроводу та повторного використання.

По суті, покажчики функцій на основі делегатів забезпечує динамічний та ефективний спосіб зіставлення подій з відповідними обробниками. Такий підхід сприяє створенню модульної, гнучкої та відокремленої системи подій, що в кінцевому підсумку призводить до більш надійного та адаптивного ігрового досвіду в рольових іграх.

3. Пріоритетна черга

Уявимо собі хаотичну сцену у roguelike грі. Відбувається бій з важким монстром, під час якого гравець ухиляється від його атак і постійно п’є зілля для зцілення. Раптом під його ногами спрацьовує пастка, яка завдає додаткової шкоди. У цьому вихорі активності порядок обробки подій може мати вирішальне значення. Саме тут вступає в дію черга пріоритетів, яка діє як складний контролер руху для системи подій, гарантуючи, що найбільш важливі події

отримують пріоритет.

Але при цьому, стандартна черга працює за принципом «першим прийшов - першим пішов»(FIFO). Події обробляються в порядку їх додавання до черги. Хоча цей підхід працює для багатьох сценаріїв, він може стати проблематичним у динамічному середовищі, яким є зазвичай жанр roguelike. Менш термінові події, такі як підняття золотої монети, можуть бути оброблені раніше, ніж більш критичні, такі як смерть гравця або зміна рівня. Це може призвести до ситуації, коли гравець «помирає» після того, як він вже випив цілюще зілля, створюючи неприємний і нелогічний ігровий досвід.

Пріоритетна черга використовує підхід FIFO і додає остаточний рівень пріоритетності. Кожній події присвоюється значення пріоритету, коли вона додається до черги. Події з вищим пріоритетом обробляються першими, незалежно від того, коли вони були додані. Це гарантує, що критичні події, такі як смерть гравця або зміна рівня, обробляються негайно, підтримуючи послідовний і логічний потік подій в ігровому світі.

Давайте повернемося до хаотичної сцени битви. Подія «Атака» від монстра може мати високий пріоритет, гарантуючи швидке отримання ушкоджень. Подія «Використання зілля», хоч і важлива, але може мати дещо нижчий пріоритет. Однак подія «Спрацювання пастки», якщо її правильно розробити, швидше за все, матиме найвищий пріоритет. Це гарантує, що персонаж гравця отримає ушкодження від пастки до того, як почне діяти цілюще зілля, що відображає миттєвий характер впливу пастки.

Хоча базові черги пріоритетів добре працюють для багатьох сценаріїв, можна розглянути і більш складні реалізації. Наприклад, гра може мати кілька рівнів пріоритетів для різних категорій подій. Високопріоритетні події можна додатково розділити на «критичні» та «термінові» залежно від їхньої важливості. Крім того, деякі черги пріоритетів дозволяють динамічно змінювати пріоритети залежно від умов, що змінюються. Уявимо, наприклад, бій з босом, в якому здоров’я боса досягає критичного рівня, що спричиняє тимчасове підвищення пріоритету подій, пов’язаних з останніми атаками боса.

Впровадження черги пріоритетів має кілька переваг для системи подій:

Покращений ігровий процес: забезпечуючи першочергову обробку критично важливих подій, система створює більш логічний та швидкий ігровий процес. Гравці можуть реагувати впевнено, знаючи, що ігровий світ відреагує відповідно і послідовно.

Гнучкість: черги пріоритетів дозволяють тонко контролювати обробку подій. Розробники можуть адаптувати значення пріоритетів до свого конкретного ігрового дизайну, підкреслюючи важливість певних подій над іншими.

Масштабованість: у міру того, як гра ускладнюється новими типами подій, черги пріоритетів можуть легко пристосовуватися до них. Призначаючи відповідні пріоритети, система може забезпечити ефективну обробку всіх подій, незалежно від загальної складності гри.

Отже, черга пріоритетів є потужним інструментом для управління системою подій у рольовій грі. Визначаючи пріоритетність критично важливих подій, вона забезпечує плавний і логічний потік дій, що в кінцевому підсумку сприяє більш захоплюючому і захоплюючому ігровому процесу.

### 3.2 Розробка алгоритмів застосунку

Ядром будь-якої гри є її алгоритми. Це ретельно розроблені набори інструкцій, які керують кожною взаємодією, поведінкою ворогів та реакцією навколишнього середовища у створеному цифровому світі.

Як було вказано вище,  у гравця не буде можливості повноцінно рухатись по локаціях, тому все, що він зможе робити – оглядати локації та взаємодіяти з ними за рахунок кліків по інтерактивним елементам гри. Головним таким елементом будуть двері.

Для початку потрібно реалізувати процедурну генерацію рівнів. Гра матиме три фіксованих рівні заглиблення, в кожному з яких буде від п’ятнадцяти до двадцяти п’яти кімнат.

Для цього можна створити три основних класа:

1) Кімната. Клас, що представляє кімнату, з набором дверей. В ньому

міститься ідентифікатор кімнати та список дверей.

```
public class Room
{
    public int id;
    public List<Door> doors;

    public Room(int id)
    {
        this.id = id;
        doors = new List<Door>();
    }
}
```

Рисунок 3.3 Реалізація класу Room

2) Двері. Клас, що представляє двері, які з'єднують кімнати.

```
public class Door
{
    public Room leadsTo;

    public Door(Room leadsTo)
    {
        this.leadsTo = leadsTo;
    }
}
```

Рисунок 3.4 Реалізація класу Door

3) Генератор рівнів. Клас, що відповідає за генерацію всієї структури рівня. В ньому міститься три методи: GenerateLevel(генерує послідовність кімнат), CreateRoom (створює нову кімнату) та CreateDoor (створює двері між кімнатами). Важливо відмітити, що кожна кімната починається з одного дверного зв'язку, який веде до наступної кімнати і після створення всіх кімнат, до кожної з них додаватимуться додаткові двері для більшої варіативності.[13-15]

```
void GenerateLevel()
{
    rooms = new List<Room>();

    // Створення початкової кімнати
    Room startRoom = CreateRoom();
    rooms.Add(startRoom);

    // Поточна кімната для генерації
    Room currentRoom = startRoom;

    // Генерація інших кімнат
    for (int i = 1; i < numberOfRooms; i++)
    {
        Room newRoom = CreateRoom();
        rooms.Add(newRoom);

        // Створення дверей між поточною та новою кімнатами
        CreateDoor(currentRoom, newRoom);

        // Переходимо до нової кімнати
        currentRoom = newRoom;
    }
```

Рисунок 3.5 Частина реалізації GenerateLevel

Також одразу створено лічильник глузду, який має шкалу від 0 до 100. Гравець починає на верхній відмітці і поступово буде втрачати глузд, що відображатиметься на відповідній шкалі. Разом із поступовим зменшенням, гравець також буде отримувати негативні ефекти. На відмітці сто з ним нічого не відбувається. Коли рівень глузду опускається до відмітки в вісімдесят, у гравця дещо тьмяніє екран. На відмітці шістдесят гравець не може чітко розгледіти деталі кімнати, тому що починає блюритись екран, окрім цього, з’являються дивні моторошні звуки. На відмітці сорок гравець більше не може чітко розуміти, які двері перед ним, тому доводиться обирати навмання. На позначці двадцять всі негативні ефекти посилюються, а також гравця починає переслідувати тінь. І врешті-решт на відмітці нуль, гравця назавжди поглинає порожнеча і він гине.


Рисунок 3.6 Частина реалізації шкали глузду

Далі перейдемо до найважливішої частини – система генерації випадкових подій.

Для цього потрібно створити новий клас – EventManager, який відповідатиме за управління генерацією подій.

```
public class EventManager : MonoBehaviour
{
    private EventPool eventPool;
    private TriggerMechanics triggerMechanics;
    private EventImpact eventImpact;
    private ImpactScaling impactScaling;

    public void Initialize()
    {
        eventPool = new EventPool();
        triggerMechanics = new TriggerMechanics();
        eventImpact = new EventImpact();
        impactScaling = new ImpactScaling();
```

Рисунок 3.7 Реалізація класу EventManager

Для того, щоб було зручно відслідковувати які події яким чином будуть з’являтись, створено два класи перерахування: Indicator та RoomEvent. Перший відповідає за те, які двері та з якою ймовірністю можуть з’являтись і відповідно які події можуть за собою викликати. Другий відповідає за сам пул можливих подій, які підв’язуються до ідикаторів.

```
public enum Indicator
{
    None,
    Blue,
    Green,
    Red,
    Secret,
    White,
    Gray,
    Orange,
    Yellow,
    Purple
}
```

```
public enum RoomEvent
{
    EmptyRoom,
    PuzzleRoom,
    RestRoom,
    ShadowRoom,
    MusicBoxRoom,
    FogRoom,
    TrapRoom,
    ArtifactRoom,
    WishingRoom,
    MysteriousShadow,
    SecretRoom,
    EndGameRoom,
    CodeLockedRoom
}
```

Рисунок 3.8 – Перерахування індикаторів подій та самих подій

Тоді система генерації подій працюватиме наступним чином: всередині класу EventManger міститься метод GenerateRoomEvent, який визначає індикатор та подію для кожної кімнати на основі ймовірностей та заданих умов і повертає випадкову подію з переданого набору.

```
public void GenerateRoomEvent(Room room)
{
    roomCounter++;

    // Зелений індикатор з'являється кожну 8 кімнату
    if (roomCounter % 8 == 0)
    {
        room.indicator = Indicator.Green;
        room.eventType = RoomEvent.RestRoom;
    }
    else
    {
        // Генерація інших подій на основі ймовірностей
        float randValue = (float)random.NextDouble();
```

Рисунок 3.9 – Метод GenerateRandomEvent.Приклад з зеленим індикатором

Дана система дозволяє створювати велику кількість різноманітних динамічних рівнів, в якій гравець зіштовхнеться з різними подіями в залежності від того, в яку кімнату він входить[28].

### 3.3 Демонстрація роботи

Головне має вигляд, як зображено на рисунку 3.5. Всього є дві кнопки: Старт, яка запускає гру та Вихід, яка вимикає застосунок.

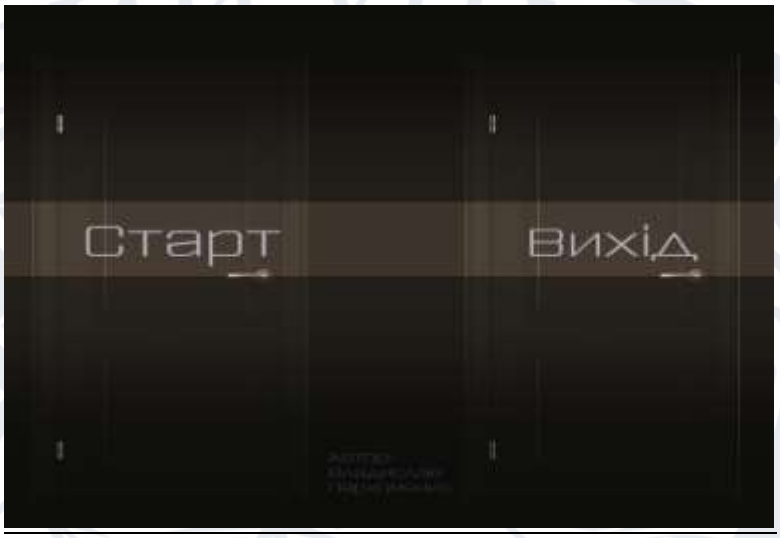


Рисунок 3.10 – Головне меню застосунку

Геймплей виглядає наступним чином: в гравця є на вибір від 2 до 5 дверей, які він може обрати і відповідно до вибору, відбудеться якась подія.

Рисунок 3.11 – Геймплей гри

На рисунку 3.5 представлено варіацію, коли у гравця є на вибір синя, червона або зелена кімнати[29].

Якщо гравець обирає зелену кімнату, він може відновити певну кількість

шкали глузду, після чого зможе перейти до наступної кімнати.

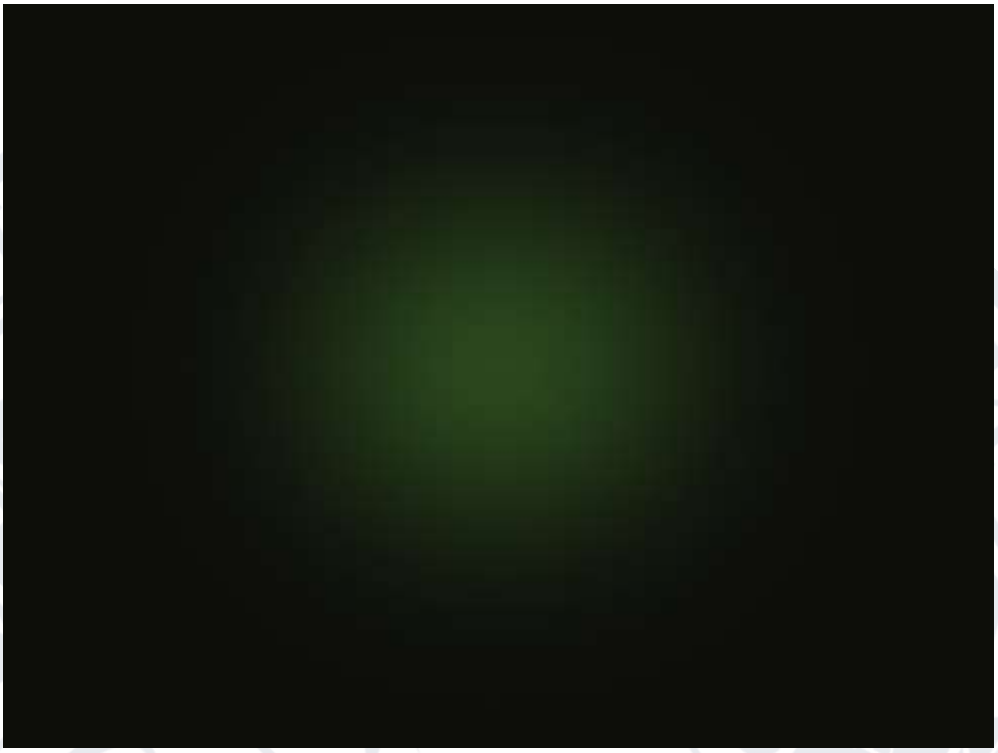

Рисунок 3.12 Приклад зеленої кімнати

Якщо гравець обирає червону кімнату, то може трапитись одна з наступних подій: гравець натрапить на пастку, гравець зустріне музичну скриньку і матиме вибір прокрутити її або ні, відповідно до чого матиме наслідки (зі скриньки вистрибне клоун з одним з трьох настроїв: веселий, що дасть цінний бонус гравцю; злий, що сильно нашкодить гравцю; сумний, що фактично не матиме суттєвого впливу на гравця) або зустріне тінь, яка зіграє з ним в рулетку програвши в якій, гравець втратить трохи глузду, а виграв ши, не втратить і зможе отримати бонуси.

Обираючи кімнати з червоним індикатором, гравець фактично ризикує власним глуздом для того, щоб потенційно отримати цінні бонуси або переваги, які допоможуть йому в подальшій грі.

Рисунок 3.13 Приклад червоної кімнати

Якщо ж гравець обирає синю кімнату, то його можуть очікувати наступні події: головоломка, пастка з можливістю ухилитись або пустка кімната.

Якщо гравець зможе пройти потрібні кімнати і вижити, або зможе виконати умови особливих подій, то в кінці його будуть очікувати білі двері, які свідчитимуть про те, що гравець пройшов гру.

Рисунок 3.14 – Фінальні двері

Важливий етап реалізації застосунку – його тестування. Провівши декілька

ітерацій, було зроблено наступні висновки:

Ітерація 1.

В першій ітерації з 10 кімнат, виявилось, що ймовірність появи деяких кімнат є занадто високою. Наприклад кімната з жовтим індикатором, яка свідчить про наявність артефактів, з’явилась перед гравцем тричі, що може розчарувати гравця через те, що він занадто легко здобуває сильні предмети і йому стає нецікаво далі досліджувати гру.

З першого тестового запуску, можна зробити висновок, що важливо чітко вибудувати баланс гри, щоб гравці могли отримувати цілісний ігровий досвід.

Ітерація 2.

В другій ітерації кількість кімнат була збільшена до 20. Виявилось, що баланс кімнат порушено не лише з кімнатами з жовтим індикатором, а ще й з фіолетовими та зеленими.

Також було виявлено неприємний баг, коли події, які повинні з’являтись, якщо гравець обрав кімнату з червоним індикатором, з’являлись в кімнатах з синім індикатором.

Ітерація 3.

Проблема великої кількості жовтих та фіолетових кімнат частково вирішена, але потрібно провести ширші тестування, щоб повністю дослідити проблеми пов’язані з балансом для покращення ігрового досвіду.

# ВИСНОВКИ

У даній роботі приведено результати розробки комп'ютерної гри жанру рогалик на основі концепції генерації випадкових подій у контексті рольових ігор. За результатами проведеного дослідження можна зробити наступні висновки:

1. Дослідження еволюції жанру рогалик в ігровому сегменті показало, що основні ідеї закладені у розробку даних ігор вибудовуються на проходженні логічних ребусів та головоломок. Алгоритмічні рішення таких ігрових застосунків вимагають запровадження функціоналу для роботи з такими елементами як випадковість та перманентна смерть.

2. Однією із основних категорій, що покладено в концепцію ігор рогаликів є випадковість подій, реалізація яких в ігрових застосунках потребує створення та проведення гравців за розробленими алгоритмами сценаріїв з метою отримання певного ігрового досвіду.

3. Алгоритми, що використовуються для генерації випадкових подій, відіграють вирішальну роль у формуванні ігрового досвіду. Однак надзвичайно важливо дотримуватися балансу між новизною, яку вносить випадковість та справедливістю, яку очікують бачити гравці, граючи в рогалики.

4. Сучасний ринок ігрових застосунків жанру рогаликів налічує достатню кількість розробок, які створюють конкуренцію та мають свої алгоритмічні ідеї, спільним серед яких є: встановлення відносин між персонажами, динаміка гри, стратегії розвитку випадкових подій.

5. Надмірна випадковість може призвести до розчарування гравців, коли результати здаватимуться надмірно неконтрольованими. В роботі також досліджено методи зваженої генерації випадкових подій, де ймовірність певних подій коригується на основі дій гравця або його прогресу в грі. Такий підхід персоналізує випадкові елементи, зберігаючи при цьому виклик від гри і заохочуючи прийняття гравцем стратегічних рішень.

6. Серед основних тенденцій, що сформували стан сучасної ігрової

індустрії можна виділити такі: інтеграція наративу та осмислений вибір, злиття жанрів, алгоритмічний прогрес, доступність та зменшення розчарування.

7. Сформовано основні механіки, що закладені у розробку ігрового застосунку: Перманентна смерть, Процедурна генерація рівнів Лічильник часу. Шкала глузду. Система випадкових подій. Випадкові події можуть чинити різний вплив на поведінку гравця: надавати переваги, створювати труднощі або змінювати динаміку рівня.

8. Розроблено ігровий застосунок жанру рогалик із використанням алгоритму генерації випадкових подій, який використовує такі алгоритмічні рішення як-от: пошук в глибину, вказівник на функцію, черга з пріоритетом.

9. Реалізація програмних модулів ігрового застосунку проведена із використанням мови програмування C# та середовища Unity. Геймплей гри зосереджено на виборі випадкових сценаріїв проходження гри, кожен із яких має свою історію завершення.

10. В підсумку, генерації випадкових подій є ключовим елементом для підвищення реграбельності рольових ігор даного жанру. Ретельно продумавши реалізацію та збалансувавши випадковість з усталеними концепціями рольових ігор, розробники можуть створити більш захопливий та незабутній досвід для гравців.

Отримані висновки можуть бути використані для створення більш захопливих і вражаючих випадкових подій у майбутніх рольових та інших ігрових жанрах. На основі розробленого додатку, інші розробники можуть створювати складні інноваційні проєкти в форматі рольових ігор, які потенційно включатимуть або розширюватимуть систему випадкових подій.

# СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Петрова Р.В., Лаптій А.А. Дослідження псевдовипадкових ефектів у комп'ютерних іграх. Матеріали v міжнародної науково-практичної конференції «Актуальні питання сучасної науки» 2020. С. 48-49.
2. Сайт Episodic Content. URL: https://episodiccontentmag.com
3. Сайт ArsTechnica. URL: https://arstechnica.com
4. Сайт RogueBasin. URL: https://www.roguebasin.com
5. Сайт Geeks for Geeks. URL: www.geeksforgeeks.org
6. Сайт NNGroup. URL: https://www.nngroup.com/
7. Сайт GDCVault. URL: https://gdcvault.com/
8. Godot Engine. URL: https://godotengine.org/
9. Jennifer Greene, Andrew Stellman. Head First C# 4th Edition. 2021. 800 с.
10. Joseph Hocking. Unity in Action, Third Edition: Multiplatform game development in C# 3rd Edition. 2022. 416 с.
11. Бреславець В.С. Технології розробки комп'ютерних ігор: довідник модуля: «Друкарня Мадрид», 2018.162 с.
12. Bill Wagner. Effective C# (Covers C# 6.0): 50 Specific Ways to Improve Your C# (Effective Software Development Series) 3rd Edition. Addison-Wesley Professional. 2016. 288 с.
13. Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest Clifford Stein. Introduction to Algorithms. Fourth Edition. 2022. 1312 с.
14. Gayle Laakmann McDowell. Cracking the Coding Interview. 189 Programming Questions and Solutions 6th Edition. 2016. 696 с.
15. Steven Skiena. The Algorithm Design Manual (Texts in Computer Science) 3rd. 2020. 810 с.
16. Пархоменко В.В., Потапова Н.А. Використання технології Nanite в UE5. III Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених. 2022. URL: https://jait.donnu.edu.ua/article/view/12282
17. Itch. URL: https://itch.io/game-assets

18. C# Guide. URL: https://learn.microsoft.com/en-us/dotnet/csharp/

19. C# Corner. URL: https://www.c-sharpcorner.com/

20. Сайт з документацією Unity. URL: https://docs.unity3d.com/

21. Palleter. URL: https://paletter.beloweb.name/sire-retro/

22. Stack Overflow. URL: https://stackoverflow.com/

23. Donald E. Knuth. Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd Edition. 1997. 672 с.

24. Noor Shaker , Julian Togelius, Mark J. Nelson. Procedural Content Generation in Games. 2016. 237 с.

25. Ністрем Роберт. Патерни програмування ігор. 2014. 432 с.

26. Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki, and Sébastien Hillaire. Real-Time Rendering, Fourth Edition. 2018. 1178 с.

27. Aditya Bhargava. Grokking Algorithms. Second Edition 2nd Edition. 2024. 320 с.

28. Steve Dahlskog, Julian Togelius. Procedural Content Generation Using Patterns as Objectives. 2014. с. 3-8 URL: https://www.researchgate.net/publication/260475455_Procedural_Content_Generation_Using_Patterns_as_Objectives

29. Дуров О. В. Ігровий застосунок в жанрі roguelike на рушії Unity. 2023. URL: https://krs.chmnu.edu.ua/jspui/handle/123456789/2961

30. Unity Asset Store. URL: https://assetstore.unity.com

31. Грудзинський Ю.Є. Алгоритми та структури даних. Навчальний посібник. 2022. С. 25-38. URL: https://ela.kpi.ua/server/api/core/bitstreams/0db974f9-16fa-459c-9f19-fab0021222ed/content

32. Габрусєв В.Ю., Вельгач А.В., Кулянда О.О. Дослідження функціональних особливостей рушія unity 3d на прикладі реалізації 3d міні-гри. 2020. С. 1-8. URL: https://sj.udu.edu.ua/index.php/kosn/article/view/1042

33. Денисенко Ю.О. Розробка ігрового застосунку з використанням системи Unity С. 2023. 26-28. URL: https://dspace.znu.edu.ua/jspui/handle/12345/12538

34. Петрова Р.В., Лаптій А.А. Дослідження псевдовипадкових ефектів у комп'ютерних іграх. Матеріали v міжнародної науково-практичної конференції

«Актуальні питання сучасної науки» 2020. С. 48-49.

35. Долгий А.І., Новіков Ю.С. Оптимізація випадкової генерації Roguelike рівнів. 2024. С. 220-225. URL: https://archive.logos-science.com/index.php/conference-proceedings/article/download/1648/1683

36. Степаненко Е. П. Дослідження методів створення віртуальних світів у відеоіграх. 2020. URL: https://openarchive.nure.ua/entities/publication/a95a9655-403f-4b4b-a689-2052847c1c4e

37. GameDev.net. URL: https://www.gamedev.net/

38. Proc Jam. URL: https://itch.io/jam/procjam

39. MIT OpenCourseware. URL: https://ocw.mit.edu/

40. Freesound.URL: https://freesound.org/

**ДОДАТОК А**

**ДЕКЛАРАЦІЯ**
про дотримання академічної доброчесності

Я, _____
_____
*Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)*

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

**ЗОБОВ'ЯЗУЮСЬ:**

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

**ПІДТВЕРДЖУЮ:**

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

**УСВІДОМЛЮЮ:**

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____
(дата) (підпис)