Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
(повна назва)
Кафедра Програмної інженерії
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження методів створення віртуальних світів у відеоіграх
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-1
Степаненко Е.П.
(прізвище, ініціали)

спеціальності 121 – Інженерія програмного забезпечення
(код і повна назва напряму)

Освітньо-наукової програми
(тип програми)
Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник к.т.н. Каук В.І.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри, проф. _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____ 20 __ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Степаненко Евеліні Павлівні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів створення віртуальних світів у відеоіграх затверджена наказом університету від "27" 03 2020 року № 473 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 8 травня 2020 р.
3. Вихідні дані до роботи методи створення віртуальних світів у відеоіграх, методи та основні поняття левел-дизайну, методи автоматизації створення віртуальних світів.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз проблемної галузі і постановка задачі, огляд методів створення віртуальних світів, понять левел-дизайну та методів автоматизації створення віртуальних світів.

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	12 березня 2020 р.	
2.	Огляд існуючих методів	20 березня 2020 р.	
3.	Методи створення віртуальних світів у відеоіграх	22 березня 2020 р.	
4.	Підготовка пояснювальної записки	27 квітня 2020 р.	
5.	Підготовка презентації та доповіді	29 квітня 2020 р.	
6.	Нормоконтроль, рецензування	9 травня 2020 р.	
7.	Занесення диплома в електронний архів	13 травня 2020 р.	
8.	Попередній захист	14 травня 2020 р.	
9.	Допуск до захисту у зав. кафедри	15 травня 2020 р.	

Дата видачі завдання 12 березня 2020 р.

Студент _____

Керівник роботи _____ к.т.н. Каук В.І.

РЕФЕРАТ / ABSTRACT

Атестаційна робота містить: 51 с., 10 рис., 1 табл., 11 джерел.

ВІРТУАЛЬНИЙ СВІТ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ЛЕВЕЛ-ДИЗАЙН, ЛАНЦЮГИ МАРКОВА, АВТОМАТИЧНА ГЕНЕРАЦІЯ ІГРОВОГО ОТОЧЕННЯ.

Мета роботи полягає у дослідженні методів створення віртуальних світів у відеоіграх, розвитку навичок з левел-дизайну, а також аналізу та створення алгоритмів генерації віртуальних світів.

В результаті роботи було проаналізовано учбову літературу з відповідної теми, а також статті та дослідження. Було розвинуто навички левел-дизайну, проаналізовано та вивчено декілька методів автоматичної генерації віртуальних світів, у тому числі процедурної генерації ігрового контенту.

VIRTUAL WORLD, PROCEDURAL GENERATION, LEVEL-DESIGN, MARKOV CHAINS, AUTOMATIC GENERATION OF GAME WORLD.

The main goal of this work is in the researching of methods of virtual worlds creation in the videogames, increasing the level-design skill, so as analyzing and development of virtual worlds generation algorithms.

As a result of this work were analyzed the education literature of the corresponding theme, so as the articles and researches. Were increased the level-design skills, analyzed and learned several methods of the automatic generation of the virtual worlds including procedural content generation.

ЗМІСТ

ВСТУП

Останнім часом ігри стають дедалі популярнішими, зокрема, через широку доступність та навіть деколи необхідність наявності мобільних гаджетів та персональних комп’ютерів, дедалі більшої популярності набувають зокрема відеоігри. Відеоігрова індустрія велика та вражає різноманітністю жанрів, сюжетів та ігрових механік. У останні роки з’являється більше інді-студій, що не мають великих бюджетів та багатих спонсорів, проте вони і не скуті рамками трендів та не соромляться робити зухвалі продукти.

У більшості ігор є віртуальний світ, що є мапою чи деяким набором мап, на яких відбувається ігровий процес. Це чи не найважливіша складова гри, що одразу привертає увагу гравця та напряму впливає на його зацікавленість у грі [1]. Створення ігрових мап це дуже відповідальний та важливий процес для будь-якої гри, що робить актуальною задачу дослідження методів створення ігрових світів.

Загалом створення ігрових мап можна поділити на ручне та автоматичне. Ручне створення мап дозволяє створювати гарно продумані локації, це особливо актуально за наявності на локації головоломок, до яких гравця потрібно розумно підвести чи сильної сюжетної складової гри, що потребує конкретних мап та/або подання певної інформації чи виникнення певних подій у чітко обумовленій послідовності.

Проте створені вручну мапи мають і кілька недоліків, зокрема, на їх створення зазвичай потрібно багато часу. Також створені вручну мапи мають менший простір для реграбельності, тобто повторного проходження гри, адже їх кількість завжди кінцева. До того ж не для кожного типу гри підходить вручну створена мапа, адже деякі ігри розраховані на те, що гравець повинен розвідувати та/або пристосовуватись до оточуючого середовища. Багато з цих ігор є багатокористувацькими, для яких наявність якомога більш різноманітних карт є критичною, адже, якщо один з гравців вже грав на якійсь мапі та знає її переваги та

недоліки, це надає йому конкурентну перевагу, що є абсолютно нечесним відносно тих гравців, що на цій мапі не грали [2].

У такому випадку краще робити якомога більше найрізноманітніших карт, проте, якщо їх ручна розробка потребуватиме багато часу та ресурсів, до того ж, все одно залишається ймовірність повторного проходження однієї і тієї ж самої мапи.

Усі вищезазначені проблеми дозволяє вирішити автоматичне генерування ігрових мап, адже написаний один раз алгоритм може генерувати нескінченну кількість несхожих одна на одну мап. Це займає набагато менше часу, аніж ручне створення великої кількості мап, та робить гру більш реграбельною через елемент неочікуванності та новизни при повторному проходженні гри [3]. Недоліками такого підходу до створення ігрових світів є ускладнена задача балансування порівняно із ручним створенням мап та складність лінійної подачі сюжету. Отже, таких спосіб не підходить для більшості сюжетних ігор, проте добре підходить для ігор, орієнтовних на реграбельність та таких, де добре вписується рандомізація.

Зазвичай задача балансування автоматичного генерування ігрових світів вимагає не лише високого рівня компетенції у програмуванні та розробці застосунків, а й у левел-дизайні локацій та балансуванні. Більш того, часто ці знання часто мають бути більш глибинними, адже балансування в умовах автоматичного генерування мають бути більш філігранними.

Тож, у атестаційній роботі досліджуватимуться питання та проблеми не лише ручного створення віртуальних світів, тобто, левел-дизайну загалом, а й автоматичного створення ігрових мап.

1 ОГЛЯД МЕТОДІВ СТВОРЕННЯ ВІРТУАЛЬНИХ СВІТІВ

1.1 Аналіз предметної галузі

Створення комп'ютерної гри є важким комплексним процесом, що часто вимагає участі команди розробників, що має складатись зі спеціалістів у різних областях.

Три базових елементи є основою розробки будь-якої гри. Перший з них – це ігрова концепція, що обумовлює увесь процес взаємодії гравця із світом гри. Її створенням займається геймдизайнер. У процесі розробки ігрової концепції він відповідає на такі питання, як:

– як гравець взаємодіятиме із ігровим світом;

– як він бачитиме ігровий світ та як буде закріплена камера;

– які істоти/предмети/елементи ландшафту траплятимуться героєві та як він буде з ними взаємодіяти;

– який досвід ми хочемо дати відчути гравцеві та ін.

Другим елементом є технологічна база для реалізації гри. Основним її інструментом є графічний рушій, адже саме він визначає набір можливостей щодо відображення та керування різними елементами гри, доступні платформи, на яких можна випустити гру та мови програмування, на яких можна буде її програмувати, які, у свою чергу, разом із рушієм, визначатимуть можливості щодо оптимізації та розмірів гри. Іноді великі компанії роблять власні рушії для великих ААА ігор – такі рушії зазвичай заточені під конкретні задачі конкретної гри. Це також дає розробникам можливість швидко виправити помилки рушія, адже у них є доступ до його вихідного коду. Як можна було здогадатись, технологічною базою гри займаються саме програмісти.

Останнім за номером, але не за значенням елементом, є візуальний стиль гри, адже саме він визначає її вигляд. Над цим зазвичай працюють митці різних профілів комп'ютерної графіки. Художники намалюють малюнки-концепти вигляду світу гри, її персонажів та навіть інтерфейсу. 3D-моделери можуть зробити за цими

референсами віртуальні трьохвимірні моделі, а аніматори – задати цим моделям певні рухи, що не лише оживить персонажів та навколишній світ, а й відобразить характер героїв чи передасть атмосферу світу.

Усі три елементи разом націлені на створення ігрового процесу, що зроблений для розважання гравця, тобто геймплею. Саме геймплей є характерною рисою ігор, що відрізняють їх від кіно або книг.

Зазвичай геймплей має два основні типи – це основний геймплей гри та геймплей на рівнях. Основний геймплей уособлює базові правила та механіки гри, що доступні гравцеві із самого її початку – персонажів та їхні характеристики, базові здібності, взаємодія одне з одним та/або із зовнішнім середовищем. Це є базою для створення рівнів та геймплею для них – дизайнеру рівнів потрібно створити цікавий простір для гравця, де він зміг би застосувати усі базові механіки, оперуючи такими речами, як простір, об’єкти та налаштування ігрового світу.

Дизайн рівнів є центральним елементом комп’ютерної гри, що поєднує безліч різних аспектів, як то архітектура, геймплей, освітлення, спецефекти, звук та багато іншого. Поєднання всіх цих аспектів робить задачу дизайну рівня такою складною, але, водночас, цікавою. Цей фактор також ускладнює завдання у разі автоматизації цього процесу.

1.2 Огляд застосунків із процедурною генерацією світу

Загалом немає застосунків, які б генерували суто ігрові мапи, це завжди ігри, адже кожна конкретна гра накладає свої унікальні обмеження на генерування ігрового світу. Ігри зазвичай не мають відкритого вихідного коду, тож неможливо точно знати які алгоритми використовуються при генерації ігрових світів у них. Можна лише проаналізувати приклади згенерованих мап та порівняти їх із загальними підходами до генерації ігрових світів.

На прикладі ігор серії Sid Meier’s Civilization можемо проаналізувати приклад генерування мапи з різними типами ландшафтів – у цих іграх мапи зазвичай генеруються окремо для кожної ігрової сесії. В залежності від заданих на початку гри параметрів, змінюються і правила генерування ландшафту. Деякі з них очевидні – кількість островів, величина материків, наявність річок та озер. Деякі менш очевидні – так, середня температура планети впливає на те, які кліматичні зони будуть доступні при генерування та яких з них буде більше, вік планети впливає на кількість на ній гір. Така генерація може відбуватись із використанням, наприклад, ланцюгів Маркова – кінцевого автомата, де кожний наступний крок залежить лише від поточного стану системи.

У більш 3D-орієнтованих іграх часто важливо правильно спроектувати мапу висот. Часто для цієї задачі використовують шум Перлина – це градієнтний шум, що складається з набору псевдовипадкових одиничних векторів (напрямів градієнта), розташованих в певних точках простору і інтерпольованих функцією згладжування між цими точками. Такий принцип генерації ландшафту може використовуватись у грі No Man’s Sky у генерації планет.

1.3 Постановка задачі

Метою цієї роботи є дослідити способи створення віртуальних світів у відеоіграх та спроектувати власний алгоритм створення віртуальних світів. Для цього потрібно виконати наступні задачі:

– провести аналіз існуючих методів створення віртуальних світів;

– дослідити принципи ручного створення віртуальних світів, адже це допоможе правильно автоматизувати процес їх створення;

– дослідити принципи динамічного генерування віртуальних світів;

– програмно реалізувати алгоритм динамічного генерування ігрового світу.

2 ПРИНЦИПИ РУЧНОГО СТВОРЕННЯ ВІРТУАЛЬНИХ СВІТІВ

Основним завданням створення віртуального світу є створення цікавої для гравця локації, що забезпечує функціонування ігрових механік. Правильне планування дозволяє сформувати чітке бачення майбутнього рівня. Саме на цьому етапі закладаються основи того, як рівень виглядатиме та гратиметься, з яких деталей він складатиметься, а також який об'єм роботи необхідно буде виконати.

Отже, перший етап планування рівня – це концептуалізація, тобто пошук відповідного концепту, ідеї, що відображує не лише цікаву локацію, а й цікавий геймплей. Дизайнер рівнів ретельно обмірковує усі особливості локації, що приверне увагу гравця та зроблять процес гри цікавим та унікальним.

При концептуалізації виникає безліч питань до майбутнього рівня. Потрібно вирішити яку ігрову локацію обрати; за що гравець запам'ятає рівень; що стане його «родзинкою»; чим приголомшить та здивує; який новий геймплейний досвід запропонує рівень гравцеві; як він гратиметься; чи можливо реалізувати ідею враховуючи обмеження обраного графічного рушію, а також чи достатньо в наявності виробничих ресурсів, аби створити цей рівень у потрібні терміни.

Під час продумування концепту важливо також подумати про грамотне наповнення контентом рівня, аби гравцю не було нудно частину рівня а частину він навпаки був перевантажений ворогами, інформацією, лутом та інше.

Візуалізація контенту – це другий крок планування ігрового рівня. Дизайнер рівнів разом із художником ігрового середовища проводять детальне дослідження вибраної тематики. Для ілюстрації ідеї формується добірка фото- та відеоматеріалу.

Мета дизайнера рівнів – пошук ключових елементів, що допоможуть у створенні унікального процесу гри. Такими елементами можуть стати незвична архітектура та планування будівель, а також специфічні особливості ландшафту.

Наприклад, якщо геймплей зосереджений на взаємодії гравця із опонентом у ближньому бою, то для цього краще підійдуть закриті приміщення (інтер'єри будівель, печери, вузькі вулички). У випадку, коли основне завдання – побудувати

рівень для снайперів, краще за все звернути увагу на відкритий простір (довгі та широкі вулиці, мости, дахи будівель, пустельні ландшафти із невеликою кількістю схованок та інше).

Важливо пам’ятати, що усі ключові елементи локації із їхнім різнобарв’ям архітектурних форм повинні допомогти у створенні необхідного саме у цій грі та саме на цьому рівні геймплею.

Після візуалізації концепту та визначення ключових моментів, з яких складатиметься локація, настає черга третього кроку – безпосереднього планування геймплею на рівні. На цьому етапі дизайнер малює схематичний план та пише документ, що загалом описує ігровий процес.

Рисунок 2.1 – Приклад широкої просторої вулиці з гри Quake 2

Далі потрібно подбати про організацію руху. Грамотно організований рух на рівні - це одна з ключових складових у створенні захоплюючого ігрового досвіду і цікавого процесу дослідження. Саме від того, як гладко буде протікати переміщення гравця від однієї локації до іншої, залежить загальне враження від гри. Мало кому сподобається слідувати маршрутом, рух яким позбавлений будь-якої логіки. Тому ще на етапі планування слід подумати про те, як зробити рух

гравця по локації максимально цікавим, комфортним і раціональним з точки зору дослідження.

Насамперед, цього можна досягти, використовуючи перешкоди та обхідні шляхи. Перешкоди і обхідні шляхи є ключовими елементами, якими організуються рух за класичним лінійним рівнем. Зверніть увагу на наступний знак - при всій своїй абсурдності в реальному житті, він дуже добре ілюструє принцип побудови лінійного руху в комп'ютерних іграх. Кінцева мета видна гравцеві відразу, але, щоб до неї дістатися, потрібно подолати численні перешкоди і використовувати обхідні шляхи.

Іншим важливим фактором є послідовність у дослідженні. Коли гравець цілеспрямовано досліджує оточення, то найменше йому хочеться пропустити який-небудь цінний предмет або цікаве місце на рівні. Тому при плануванні руху потрібно обов'язково пам'ятати про принцип послідовності. Послідовний рух можна порівняти з потоком води. Подібно воді, що тече по руслу річки, гравець оглядає кімнату за кімнатою і поступово просувається вперед до своєї кінцевої мети - виходу з рівня. При правильній організації кінцева мета завжди повинна знаходитися останньої в ланцюжку приміщень для дослідження, тому що гравець може просто вийти з локації завчасно, пропустивши цілий шматок рівня.

Для організації одностороннього руху в дизайні рівнів використовують метод відсікання попередньої локації за рахунок перепаду висот. Його сенс полягає в створенні височини, стрибаючи з якою гравець не може повернутися назад. В результаті ми не залишаємо гравцеві вибору, окрім як рухатися вперед. Цей прийом також допоможе замкнути гравця на певній ділянці рівня, який стане ареною для сутички з безліччю ворогів або битвою з фінальним босом.

Окрім класичного прийому з перепадом висот, для відсікання попередньої локації також використовують будь-які інші об'єкти, що блокують рух гравця назад (зачинилися двері, раптово утворилися завали, турнікети та інше).

Для багатьох ігор з відкритим світом характерна наявність локацій, рух якими побудовано у вигляді петлі. Як правило, гравець проходить через локацію до точки виконання завдання і потім відразу ж виходить в самий початок через

прохід, що відкрився, або просто зістрибнувши з височини. Цей прийом дозволяє оптимізувати маршрут і позбавити гравця від довгого шляху назад по місцевості, на якій нічого не відбувається, тому що всі вороги були тільки що знищені, приміщення досліджені, а корисні предмети зібрані.

Рисунок 2.2 – Приклад прийому перепаду висот у Left 4 Dead

Дуже часто дизайнери рівнів надають гравцеві можливість відвідати одну й ту ж локацію кілька разів. Головною метою майже завжди є перевикористання ігрового оточення для економії ресурсів і збільшення часу гри. Адже в одних і тих же декораціях можна дуже легко розповісти відразу кілька історій, показавши, як зовнішній вигляд локації і активність на ній змінюються під впливом будь-яких подій. Зазвичай рух через один і той же рівень будується кілька разів і з кожним разом гравцеві пропонуються нові геймплейні сценарії.

Іноді для адаптації перевикористаного оточення під нові ігрові механіки одного лише додавання ворогів мало. Для цього нам буде потрібна особлива глобальна подія, яке повністю змінить умови гри і дозволить поглянути на локацію під іншим кутом.

Тупики, які в підсумку виявляються єдиним правильним шляхом є ще одним ефективним способом організації руху. На практиці це виглядає так - гравець

впирається в глухий кут і починає уважно оглядатися навколо. Нічого не знаходячи, він засмучується і обертається на 180 градусів. Ось тут йому в очі і кидається шлях далі. Таким чином, правильний шлях помітний тільки якщо зайти в глухий кут і подивитися звідти в сторону, звідки щойно прийшов.

При плануванні руху дуже важливо наочно показати, куди можна йти, а куди не можна. Для цього потрібно чітко позначити межі рівня, щоб заощадити час гравця на спробах потрапити туди, куди спочатку пройти неможливо. При цьому важливо уникати будь проміжних станів - якщо дорога закрита, то це повинно бути очевидно з першого погляду.

Найпопулярнішим способом показати, що сходи заблоковані, є створення непрохідних барикад з якогось сміття або меблів. Теж саме стосується і дверей. Якщо розмістити кілька об'єктів перед замкненими дверима, які візуально заблокують прохід, то гравцеві відразу стане очевидно, що це глухий кут. У той же час, прочинені всередину двері виглядають як запрошення і викликають бажання підійти і досліджувати простір за ними.

Рисунок 2.3 – Приклад створення перепони у грі Hitman

Для управління рухом гравця можна також використовувати так зване «негативний» простір. Як правило, «негативним» називають такий простір, в якому

не хочеться перебувати в силу наявності будь-якої небезпеки (відкритий порожній простір без схованок, небезпечні вороги та інше).

Наступна важлива річ, про яку слід подбати – це заохочення дослідження. Напевно, мало хто відмовиться відчути себе першопрохідцем, що досліджує загублені світи. Прагнення людей дізнатися і побачити щось нове зробило процес дослідження найпопулярнішою активністю в сучасних іграх.

В основі цікавого процесу дослідження лежить принцип заохочення. Гравець повинен бути завжди впевнений у тому, що куди б він не забрався, його цікавість буде обов'язково будуть винагороджена чимось новим. Далі буде розглянуто найпопулярніші способи заохотити гравця. До першого типу нагород відносяться будь-які цінні предмети – зброю, боєприпаси, аптечки, броня, ключі від дверей, скарби, ресурси і т. д.

Ховати предмети, що підбираються, таким чином, що вони стають помітні, тільки якщо подивитися з глухого кута в зворотному напрямку – це ще один гарний спосіб заохотити гравця за вивчення кожного закутка.

До другого типу нагород відносять нові геймплейні можливості, що дають гравцеві якусь перевагу. Як правило, це альтернативні маршрути, короткі шляхи до мети, точки огляду, нові способи усунення супротивника і т. д.

Рисунок 2.4 – Приклад неочікувано знайденого скарба з гри Uncharted 4

До третього типу нагород відносять елементи сюжетного оповідання – скриптові сцени за участю персонажів, подача сюжету за допомогою ігрового оточення, діалоги, текстові записки і т. д. Деякі ігри із продуманою історією віртуального світу та/чи сюжетом заохочуватимуть гравця на ретельне дослідження кожного закуточка гри саме для того, аби знайти якомога більше інформації. Наприклад, частина історії ігрового всесвіту Dark Souls подається саме через опис деяких предметів, що мотивує гравця на дослідження ігрового світу заради пошуку предметів.

Рисунок 2.5 – Приклад подання сюжету через опис предметів у грі Dark Souls 2

До четвертого типу нагород відносять різні приколи і пасхалки. Як правило, вони потрібні, аби розважити гравця і не несуть великого сенсового навантаження. Часто такі пасхалки мають посилання на популярні твори масової культури, меми або на інші ігри автора чи ігрового видавництва.

Для того, щоб заманити покупця всередину магазину, зазвичай йому демонструють товар на вітрині. У дизайні рівнів існує схожа практика, сенс якої

полягає в залученні уваги гравця цінних предметом, розташованим за якою-небудь перешкодою. Для гравця поворушити мізками і знайти вхід до схованки з бонусами завжди набагато приємніше, аніж просто підібрати предмет з підлоги. Щойно гравець бачить приманку у вигляді корисного предмета, у нього відразу ж виникає цілком природнє бажання уважніше оглянути все навколо і знайти спосіб потрапити всередину.

Досяжні орієнтири, які видно здалеку це ще один вид приманки, мотивуючий гравця на дослідження. Тут дуже важливо розуміти, що об'єкт, який виступає в ролі точки тяжіння (будівля, яку всі знають, монументальна споруда, елемент ландшафту), обов'язково повинен бути досяжним. Простіше кажучи, якщо гравець бачить на горизонті хмарочос, то він повинен завжди мати можливість до нього дістатися і потрапити всередину.

Наявність схованок і скарбів – це ще один потужний спосіб мотивувати гравця на дослідження оточення. Для цього дизайнеру досить лише натякнути на існування прихованих місць через внутрішньоігрові записки, діалоги персонажів, карти скарбів і т. д.

Ще один цікавий вид дослідницької активності в іграх пов'язаний з пошуком підказок з кодами до сейфів, які за традицією зберігають в собі найцінніші предмети.

Процес дослідження ігрового світу стане ще цікавіше, якщо забезпечити варіативність шляхів між локаціями. Бажання дізнатися, куди виведе та чи інша доріжка, тільки підсилює цікавість гравця.

Також дуже важливо наситити простір контентом. Навіть невеликий ігровий світ може на довгі години затягнути в себе гравця, якщо забезпечити йому високу щільність ігрового контенту.

Простота взаємодії з оточенням - це один з ключових факторов у створенні комфортного ігрового досвіду. Якщо гравець не може швидко визначити, який з елементів оточення є інтерактивним, то це обов'язково викличе у нього роздратування. Якраз для вирішення такого роду проблем в дизайні рівнів існує поняття «візуальної мови».

Візуальна мова – це засіб комунікації між гравцем і елементами оточення, що дозволяє за допомогою системи візуальних підказок (минаючи призначений для користувача інтерфейс) передати інформацію про стан та спосіб взаємодії з об'єктом.

Принцип роботи візуальної мови ґрунтується на створенні у гравця стійкою асоціації між зовнішнім виглядом об'єкта і його геймплейною функцією. Визначити взаємозв'язок між зовнішнім виглядом об'єкта і його функцією можна тільки в тому випадку, якщо дотримується принцип постійності. Згідно з цим принципом, усі без винятку копії інтерактивних об'єктів на рівні повинні реагувати на взаємодію однаковим чином.

Використання візуальної мови для позначення інтерактивних елементів оточення допомагає наочно показати гравцеві його геймплейні можливості. Простіше кажучи, гравець може легко «читати» оточення - бачити яким чином і з якими саме об'єктами він може взаємодіяти.

Візуальна мова не лише допомагає гравцеві миттєво «читати» оточення і бачити свої геймплейні можливості, але і забезпечує сприятливе середовище для функціонування ігрових механік [4]. За допомогою неї можна донести до гравця геймплейну функцію об’єктів. Будь-який гравець знає, що стрілянина по червоній бочці обов'язково спричинить за собою її вибух. Маркування вибухонебезпечних об'єктів червоним кольором – це найпростіший спосіб натякнути гравцеві на їх геймплейну функцію.

Також за допомогою візуальної мови можна передати статус об’єкта. Так, зачинені не інтерактивні двері часто закривають решіткою, в той час, як такі, якими можна скористатись, залишаються без решіток. Також за допомогою візуальної мови можна підкреслити взаємозв’язок об’єктів, з’єднавши їх дротом; натякнути на близькість схованки спеціальним символом десь неподалік, натякнути на пастку та інше.

Важливо дотримуватись принципу постійності. Недотримання принципу постійності обов'язково призведе до великих проблем у «читаності» ігрового

оточення. Тому, при встановленні для гравця певних правил, необхідно їх дотримуватися.

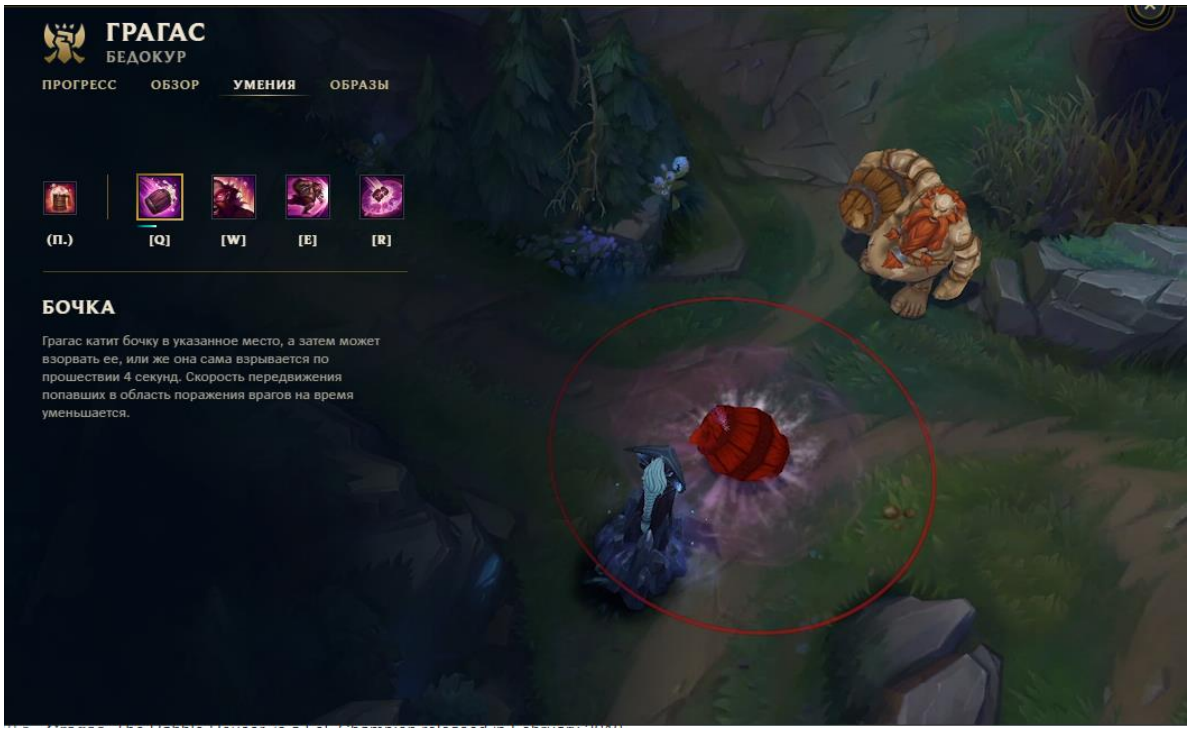

Рисунок 2.6 – Приклад візуального відмічення вибухонебезпечного об’єкту на прикладі огляду вміння чемпіона з League Of Legends

Отож, правил для створення грамотного дизайну рівнів, з яких і складатиметься дизайн віртуального світу, досить багато. Тож створення віртуального світу відеогри рівень за рівнем є досить складною, довгою та нетривіальною задачею. Часу та ресурсів на це не завжди вистачає та більш того не завжди потрібно створювати рівні власноруч – у деяких випадках їх навпаки краще генерувати.

3 ПРИНЦИПИ ДИНАМІЧНОГО ГЕНЕРУВАННЯ ВІРТУАЛЬНОГО СВІТУ

Цифрова генерація контенту (Digital Content Generation – DCG) - підмножина алгоритмів процесорного генерування вмісту (Procedural Content Generation – PCG) – є основою в сучасних відеоіграх, яка формується набором алгоритмів для створення рівнів, карт, активів, символів, зброї та іншого, пропонуючи зменшення витрат на виробництво ігор. Основна мета DCG – створення нових ігрових елементів, офлайн або в режимі реального часу.

Із самого початку розробки відеоігор у розробників виникали проблеми щодо пам'яті, що займалася створеними ними ігровими рівнями. Як правило, це вводить обмеження на розмір створених віртуальних світів. Для того, щоб мати можливість створювати більші, або навіть безмежні світи, розробники зосередилися на зменшенні розміру даних, що використовуються для створення віртуального світу. Таким чином, алгоритми DCG були створені для використання в якості інструменту для зменшення місця зберігання, зайнятого рівнями, створеними у відеоіграх.

DCG також використовується розробниками для вирішення великої проблеми в циклі розвитку відеоігор: час. Розробники використовують ці алгоритми, щоб зменшити кількість часу, необхідного для створення гри або певних її функцій.

Процедурне генерування – це процес генерування контенту алгоритмічно, а не вручну. Цей напрямок досліджень не є новим для комп’ютерних наук, де було розроблено кілька робіт. Процедурне покоління виникло в 1975 році, коли Бенуа Мандельброт визначив фрактальний об'єкт, який описує таку повторювану або самоподібну математичну закономірність. Ця ідея використовується в декількох додатках, таких як створення текстур, міста, ліси на основі L-систем [5], місцевості, відеоігри та інше. Зазвичай у відеоіграх цей процес називається динамічним генеруванням вмісту.

Як характерні приклади використання PCG називаються: генерація без участі людини підземель в пригодницьких іграх (таких як The Legend of Zelda), яка отримує новий світ на кожному запуску; система, що створює нові типи озброєнь в грі в космічному сеттингу в залежності від дій гравця; генерація повної, грабельної та збалансованої настільної гри; внутрішня процедура ігрового движка, швидко заповнює ігровий світ рослинами; інструмент, що дає можливість створювати людині карти для стратегічної гри, і при запитах і заданих змінах перераховується карту для її поліпшення, а також пропонує варіанти, що дозволяють зробити карту більш збалансованою і цікавою. У той же час, простий редактор карт, штучний інтелект для настільної гри або інструмент інтеграції створеного контенту не відносяться до PCG.

Потужність процедурного генерування дозволяє інтегрувати кілька наукових досліджень для вирішення проблем. Наприклад, у 2013 році Женева та ін. [6] визначили новий спосіб генерації рельєфу на основі гідрології. Їх алгоритм приймає в якості вхідних даних контур місцевості та деяких річок, наданий користувачем. Потім алгоритм візьме вхідну інформацію і почне генерувати повну дренажну річкову мережу, що призведе до повного утворення річок, що йдуть від джерел до виводів, з очікуваною зміною висоти. Після створення річок вони починають працювати на решті місцевості, використовуючи різні будівельні блоки та змінюючи місцевість відповідно до річок, щоб загальний результат карти був топологічно правильним.

Зокрема в області розробки відеоігор, динамічне генерування вмісту довгий час використовувалося для створення більших рівнів із меншою кількістю даних для вирішення проблем, пов’язаних із зберіганням пам’яті. Після того, як проблеми зі зберіганням пам’яті розвіялися, DCG почали використовувати для генерування інших видів контенту для збільшення ефективного діапазону контенту, який демонструвався гравцеві, уникаючи значної частини монотонності, що створюється повторюваним контентом. Ці алгоритми не можуть використовуватись під час виконання і вони можуть створювати вміст, який потрібно вбудувати в гру після її закінчення. Наприклад, ліс може бути створений

процедурно для подальшого введення в гру як статичний актив, який залишатиметься однаковим протягом усієї гри.

Можна класифікувати DCG відповідно до моменту генерації даних (онлайн проти офлайн), або пріоритетності їх залежностей у грі (необхідне проти необов'язкового), або на основі базових значень для початку генерування динамічних даних (простий насіннєвий та векторний параметри), або за рівнем випадковості (детермінований проти недетермінованого), або шляхом вимірювання еволюції поколінь (одноразова побудова проти побудови постійного часу) тощо.

Важливим фактом є те, що ці методи розробники використовували як інструменти для скорочення часу розробки проекту та збільшення кількості контенту, представленого гравцеві.

Створюваний контент повинен задовольняти певним умовам і таким чином вирішувати відповідні проблеми, і на практиці найчастіше розглядаються наступні властивості [7]:

– швидкість: в залежності від завдання вимоги різняться від мілісекунд до місяців, але в загальному випадку контент повинен бути створений вчасно для задоволення потреб ігрового процесу;

– надійність: деякі генератори створюють «купу чогось», в той час як інші можуть гарантувати виконання заданих критеріїв, наприклад завжди забезпечувати можливість прохідності гравцем до виходу лабіринту;

– контролепригодність: здатність контролювати згенерований контент виходячи з ситуації і надання геймдизайнеру відповідної волі (ця властивість далеко не завжди потрібна); наприклад, генерація гладкого довгастого каменю або створення рівня з певною атмосферою;

– різноманітність: створення такого контенту, який був би якомога різним на різних запусках, і при погляді на нього гравець не відчував одноманітність;

– креативність і правдоподібність: генерація такого контенту, який виглядає так, як ніби його створила людина, а не генератор.

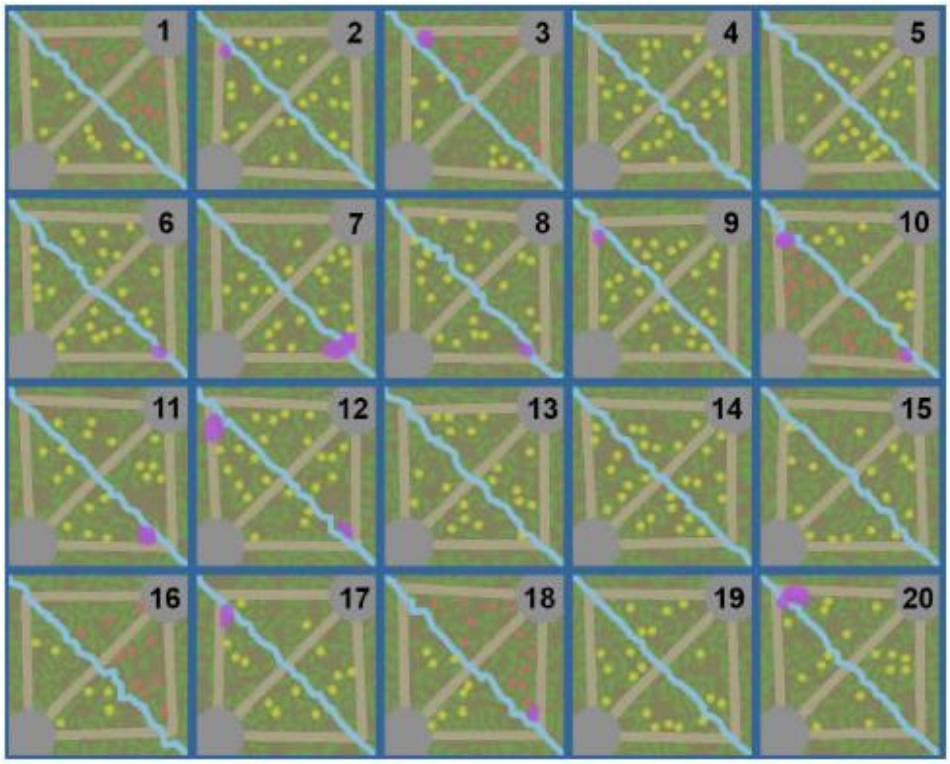

Рисунок 3.1 – Приклад процедурно згенерованих карт для MOBA

Чудовий приклад цього був представлений в 2011 році Брауном [8], розробивши інструмент, який передбачає глибоку інтеграцію процесуального генерування вмісту в механіку і естетику ігор для багатьох жанрів, включаючи ігри на основі PCG [9].

Для грамотної реалізації алгоритму процедурної генерації рівня потрібно чітко визначити усі можливі види ландшафту, що будуть присутні на цьому рівні та як вони впливатимуть на гравця та яка ймовірність генерації того чи іншого виду ландшафту по сусідству. Якщо це шматок відкритого світу на поверхні планети, то, залежно від жанру гри, там зможуть знаходитись такі типи ландшафту, як рівнина, ліс, пагорби, гори, піски, річка або інша водойма, лава, слиз, метал, камінна кладка, асфальт, крига та інше. Наприклад, по рівнині або асфальту герой рухатиметься швидко; крига, метал або слиз будуть заважати йому рухатись, адже такі поверхні доволі слизькі; по крутим горам або лаві герой може і не змогти рухатись. Ймовірність генерації тайлу (тобто шматка локації) типу лід разом із тайлом типу піску буде нульовою, у той час, як ймовірність генерації тайлу типу річки разом із

тайлом типу рівнини можна зробити близькою до значення у, наприклад, 10 – 20%, а поряд із іншим тайлом типу річки – біля 40 – 50%.

Гарним прикладом випадкової генерації ландшафту є серія ігор Civilization за авторством Сіда Мейера. Зокрема, у Civilization V можна почати гру на процедурно згенерованій карті та навіть обрати тип карти (із одним великим материком, купкою маленьких, або ж навіть світ із купи островів) та обрати такі параметри генерації, як кількість гір, клімат планети, кількість річок та озер та інше. І у рамках такої гри, як Civilization це цілком виправдано, тому що це стратегічна гра із високим рівнем реграбельності та можливістю мультиплеєру, тож критично важливим для цієї гри є елемент неочікуванності – коли гравець не знає точно яких розмірів земля, де він опинився, хто та що навколо, які ресурси навколо та у який бік вигідніше досліджувати місцевість, у якому місці з'явились поселенці інших держав та де розташовані міста-держави. Такого роду знання могло б дати гравцеві стратегічну перевагу, що з одного боку допомогло б йому перемогти у мультиплеєрі, а з іншого боку йому вже було б не цікаво грати з супротивником штучним інтелектом.

Рисунок 3.2 – Приклад процедурно згенерованої карти у грі Civilization V

Окрім цих моментів повернемось до більш базових. Зокрема, потрібно продумати та враховувати як мінімум такі пункти:

а) кожен тайл або їх сукупність, якими можна пройти, повинні бути досяжними;

б) контент має бути якомога більш рівномірно розташований на рівні, тобто не має бути якогось скупчення пасток, схованок, босів та винагород серед напівпустого у плані контенту рівня;

в) характеристики об’єктів на рівні (а особливо – монстрів, з якими гравцеві доведеться битись та луту, що з них випадатиме або ж просто знаходитиметься на рівні) повинні підлаштовуватись під поточний рівень та силу героя гравця;

г) гравцеві завжди має бути цікаво проходити гру, тож потрібно особливо уважно подбати про те, аби контент (монстри, лут, не ігрові персонажі та інше) був присутній та був різноманітний, а також інтерактивний; щоб його не було забагато чи навпаки замало.

Розглянемо приклад процедурної генерації світів у грі No Man’s Sky. У цій грі не дотримались останнього правила, тож на момент релізу гравцям у цю гру було досить нудно грати. Всі планети були схожі одна на одну, не були достатньо інтерактивними, а переліт займав досить багато часу, тож гравцям часто просто набридало грати у цю гру. До того ж не було конкретної глобальної кінцевої мети гри.

Як пишуть у багатьох відгуках, якщо нікуди не поспішати і затриматись надовго на одній планеті, то її одноманітні пейзажі швидко набриднуть. Місцевості навіть на протилежних кінцях окремо взятого глобуса мало чим відрізняються, «точки інтересу» понатикані на кожному кроці, а складання повного каталогу живого світу наганяє зовсім вже дику нудьгу [10].

Аби подібного не траплялось важливо не лише задати правильні параметри перед початком генерації, але й ретельно протестувати її, аби, якщо це потрібно, підправити баланс, тобто ймовірності виникнення тих чи інших об’єктів у тих чи інших умовах.

Рисунок 3.3 – Приклад дослідження планети у грі No Man’s Sky

Проте, завжди потрібно бути готовим до правок ймовірностей, додавання нових видів ландшафту та/або об’єктів та нового кола повторних тестувань та правок балансу та ймовірностей. Проте, такий об’єм робіт все ще менший, аніж у ручного створення рівнів.

3.1 Розробка плану алгоритму процедурної генерації ігрового світу

Алгоритм генерування рівня складатиметься з кількох кроків. Спочатку згенеруємо мапу, що складатиметься із сукупності тайлів різного типу ландшафту. На другому розташуємо на локації ігрові об’єкти різного типу, як живі, так і не живі та як гарні, так і погані для гравця – наприклад, пастки, схованки, скрині, ворожі, нейтральні або дружні неігрові персонажі.

Припустимо, що маємо спроектувати алгоритм для генерації ігрової локації типу підземелля для гри у жанрі Roguelike. Припустимо також, що маємо п’ять можливих типів ландшафту: земля, лід, лава, червоний камінь та крижаний камінь.

3.1.1 Генерування ландшафту

Насамперед маємо згенерувати локації заздалегідь заданого розміру, що складається з ділянок землі певного типу з переліку можливих типів ландшафту. Кожен з типів ландшафту має ймовірність виникнення поряд із іншими типами, включаючи самого себе.

Існує кілька підходів до генерації ландшафту. Перший – поклітинний, тобто тип кожної клітинки визначається окремо, на основі того, який тип мають клітинки навколо та відповідно які типи та з якою ймовірністю може мати ця клітинка.

Другий заснований на генерації так званого біому – одразу деякої області певного типу клітин зазвичай випадкового розміру у рамках заздалегідь заданих граничних розмірах. Такий підхід робить згенерований ландшафт менш хаотичним та виключає виникнення випадкових одиничних клітинок одного типу посеред багатьох клітинок іншого типу.

3.1.2 Використання ланцюгів Маркова у процедурній генерації ігрової мапи

Генерація ландшафту у будь-якому разі базуватиметься на генерації випадкових чисел. Проте, оскільки у нас є чіткі ймовірності переходу з одного стану в інший, доречно було б використовувати ланцюги Маркова при вирішенні цієї задачі.

Ланцюг Маркова в математиці – це випадковий процес, що задовольняє властивість Маркова і який приймає скінченну чи зліченну кількість значень (станів). Тобто це послідовність випадкових подій із кінцевою кількістю результатів, що характеризується властивістю незалежності майбутніх станів від минулого. На майбутній стан процесу впливає лише поточний стан, а не послідовність дій у минулому.

Ланцюги Маркова мають безліч застосувань як статистичних моделей процесів у реальному світі, таких як вивчення систем круїзного контролю в автотранспортних засобах , черг або ліній клієнтів, які прибувають в аеропорт, курсів обміну валют і динаміка популяції тварин.

Марківські процеси є основою для загальних методів стохастичного моделювання, відомих як ланцюг Маркова Монте-Карло , які використовуються для моделювання вибірки із складних розподілів ймовірностей, і знайшли застосування в байєсівській статистиці та штучному інтелекті. Існує два види ланцюгів Маркова – ланцюги Маркова дискретного часу та ланцюги Маркова безперервного часу.

Послідовність дискретних випадкових величин $\{X_n\}_{n\geq 0}$ називають простим ланцюгом Маркова із дискретним часом, якщо виконується умова (1):

$$\mathbb{P}(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, \dots, X_0 = i_0) = \mathbb{P}(X_{n+1} = i_{n+1} | X_n = i_n), \qquad (1)$$

де $\{X_n\}$ – область значень випадкових величин, що називається простором станів ланцюга, а номер n – номером кроку.

Матриця $P(n)$, де (2) є матрицею перехідних ймовірностей на n-му кроці, а вектор $\mathbf{p} = (p_1, p_2, \dots)^{\mathrm{T}}$, де $p_i \equiv \mathbb{P}(X_0 = i)$ – початковим розподілом Маркова:

$$P_{ij}(n) \equiv \mathbb{P}(X_{n+1} = j | X_n = i). \qquad (2)$$

Очевидно, що матриця перехідних ймовірностей є стохастичною, як показує формула (3).

$$\sum_j P_{ij}(n) = 1, \quad \forall n \epsilon \mathbb{N}. \tag{3}$$

Ланцюг Маркова є однорідною, якщо матриця перехідних ймовірностей не залежить від номеру кроку, тобто $P_{ij}(n) = P_{ij}, \ \forall n \epsilon \mathbb{N}$.

У іншому випадку ланцюг Маркова називається неоднорідним.

Сімейство дискретних випадкових величин $\{X_t\}_{t \geq 0}$ називають ланцюгом Маркова із безперервним часом тоді і тільки тоді, коли:

$$\mathbb{P}(X_{t+h} = x_{t+h} | X_s = x_s, \ 0 < s \leq t) = \mathbb{P}(X_{t+h} = x_{t+h} | X_t = x_t). \tag{4}$$

Ланцюг Маркова з безперервним часом є однорідним, якщо:

$$\mathbb{P}(X_{t+h} = x_{t+h} | X_t = x_t) = \mathbb{P}(X_h = x_h | X_0 = x_0). \tag{5}$$

Так само, як і у випадку дискретного часу, кінцевомірні розподіли однорідного ланцюга Маркова із неперервним часом повністю визначені початковим розподілом $\mathbf{p} = (p_1, p_2, \dots)^{\mathrm{T}}, \ p_i = \mathbb{P}(X_0 = i), \ i = 1, 2, \dots$ та матрицею перехідних функцій або перехідних ймовірностей:

$$\mathbf{P}(h) = \Big(P_{ij}(h)\Big) = \mathbb{P}\big((X_h = j) \big| X_0 = i\big). \tag{6}$$

Стохастичний процес має марківську властивість тоді й тільки тоді, коли умовний розподіл ймовірностей майбутніх станів процесу залежить лише від поточного стану.

Ланцюг Маркова відповідає наступним властивостям: відновлюваність, періодичність, тимчасовість і повторюваність, ергодичність, має стаціонарний розподіл та граничну поведінку, оборотність та інші. Зупинимось на них дещо докладніше.

Кажуть, що ланцюг Маркова є невідворотним, якщо є можливість потрапити до будь-якого стану з будь-якого стану. Далі пояснюється це визначення більш формально.

Кажуть, що стан j є доступним зі стану i, тобто $i \to j$, якщо система, що запущена в стані i, має ненульову ймовірність переходу в стан j в якийсь момент. Формально стан j доступний зі стану i, якщо існує ціле число $n_{ij} \geq 0$ таке, що:

$$\mathbf{Pr}\left(\left(X_{n_{ij}} = j\right) \middle| X_0 = i\right) = {p_{ij}}^{(n_{ij})} > 0. \tag{7}$$

Це ціле число може бути різним для кожної пари станів, отже, індексів n_{ij}. Якщо ж n дорівнюватиме нулю, це означатиме, що кожен стан є доступним для самого себе за визначенням. Відношення доступності є рефлексивним та транзитивним, але не обов'язково є симетричним.

Кажуть, що стан i пов'язаний зі станом j, тобто $i \leftrightarrow j$, якщо виконуються обидві умови $i \to j$ та $j \to i$. Пов'язаний клас – це максимальний набір станів C такий, що кожна пара станів C пов'язана одна із одним. Пов'язаність – це еквівалентне відношення, а комуніцюючі класи – це класи еквівалентності цього відношення. Комуніцюючий клас є також закритим тоді й тільки тоді, коли він не має вихідних стрілок на цьому графі.

Стан i називають кінцевим, якщо для всіх j таких, що $i \to j$ також вірно, що $j \to i$. Стан i є несуттєвим, якщо він не є суттєвим. Стан є кінцевим тоді й тільки тоді, коли її комуніцюючий клас закритий.

Кажуть, що ланцюг Маркова є невідворотнім, якщо його простір станів утворює єдиний комунікаційний клас, тобто якщо з будь-якого стану можна потрапили до будь-якого стану.

Ланцюг Маркова має властивість періодичності. Стан i має період k, якщо будь-яке повернення до стану i відбувається у таку кількість проміжків часу, що кратна k. Формально період стану i визначається, як:

$$k = \gcd\{n > 0 : \Pr(X_n = i | X_0 = i) > 0\}, \quad (8)$$

де gcd – це найбільший загальний дільник, та за умови, що цей набір не порожній.

Інакше період невизначений. Слід зауважити, що, хоча у стану і є період k, можливо не вдасться досягти стану у k кроків.

Наприклад, припустимо, що можна повернутись до стану у $\{6, 8, 10, 12, \ldots\}$ кроків часу; тоді k мало б значення 2, не зважаючи на те, що його немає у вищенаведеному списку.

Якщо ж $k = 1$, тоді кажуть, що стан є аперіодичним. Інакше ($k > 1$) кажуть, що стан періодичний із періодом k. Марківський ланцюг є аперіодичним, якщо кожен стан є аперіодичним. У невідновлюваному ланцюзі Маркова достатньо мати лише один аперіодичний стан, аби вважати усі стани аперіодичними. Також кожен стан двостороннього графіка має парний період.

Кажуть, що стан i є тимчасовим, якщо, враховуючи, що ми починаємо у стані i, є ненульова ймовірність того, що ми ніколи не повернемось до i. Формально нехай випадкова величина T_i є першим часом повернення до стану i (так званий «час удару»):

$$T_i = \inf\{n \geq 1 : X_n = i\}. \quad (9)$$

Кількість $f_{ii}^{(n)} = \Pr\big((T_i = n) \big| X_0 = i\big)$ –це ймовірність того, що ми повернемось до стану i вперше за n кроків. Тому стан i є тимчасовим, якщо:

$$\Pr(T_i < \infty | X_0 = i) = \sum_{n=1}^{\infty} f_{ii}^{(n)} < 1 \quad (10)$$

Стан i є періодичним (або стійким), якщо він не є тимчасовим. Гарантовано, що періодичні стани з вірогідністю 1 мають кінцевий «час удару». Повторність і швидкоплинність – це властивості класу, тобто вони або утримуються, або не мають однакового рівня для всіх членів класу, що спілкується.

Кажуть, що стан i є ергодичним, якщо він періодичний та позитивно періодичний. Іншими словами, стан i є ергодичним, якщо він є рецидивним, має період 1 і має кінцевий середній час рецидиву. Якщо всі стани в невідводному ланцюзі Маркова є ергодичними, то кажуть, що ланцюг є ергодичним.

Можна показати, що невідновлюваний ланцюг Маркова з кінцевим станом є ергодичним, якщо він має аперіодичний стан. У більш загальному сенсі, ланцюг Маркова є ергодичним, якщо існує число N таке, що будь-який стан може бути досягнуто з будь-якого іншого стану у будь-яку кількість кроків, що будаб менше або дорівнювала б N. У випадку повністю пов'язаної матриці переходу, де всі переходи мають ненульову ймовірність, ця умова виконується при $N = 1$.

Ланцюг Маркова з більш ніж одним станом і лише одним вихідним переходом на стан або не є невідмінним, або не є аперіодичним, отже, не може бути ергодичним.

Якщо ланцюг Маркова - це однорідний за часом ланцюг Маркова, так що процес описується єдиною, незалежною від часу матрицею p_{ij}, то вектор π називається стаціонарним розподілом (або інваріантною мірою), якщо $\forall j \in S$ та це задовольняє (11).

$$0 \leq \pi \leq 1; \quad \sum_{j \in S} \pi_j = 1; \quad \pi_j = \sum_{i \in S} \pi_i p_{ij}. \tag{11}$$

Невідворотній ланцюг має позитивний стаціонарний розподіл (стаціонарний розподіл такий, що $\forall i,\ \pi_i > 0$) тоді й тільки тоді, коли всі його стани поєднуються позитивно. У цьому ж випадку π є унікальним і пов'язаний з очікуваним часом повернення:

$$\pi_j = \frac{C}{M_j}, \tag{12}$$

де C – нормалізуюча константа.

Окрім цього, якщо позитивний рецидивуючий ланцюг є одночасно невідворотним і аперіодичним, вважається, що він має кінцевий розподіл; для будь-якого i та j:

$$\lim_{n\to\infty} p_{ij}^{(n)} = \frac{C}{M_j}. \tag{13}$$

Також немає припущення про початковий розподіл; ланцюг сходить до стаціонарного розподілу незалежно від того, з чого він починається. Таке π називають рівноважним розподілом ланцюга.

Якщо ланцюг має більше одного закритого комунікаційного класу, його стаціонарні розподіли не будуть унікальними (розглянемо будь-який закритий комунікаційний клас C_i у ланцюзі; кожен з них матиме своє унікальне стаціонарне розповсюдження π_i. Розширення цих розподілів на загальний ланцюг, встановлення всіх значень на нуль поза класом зв'язку, дають, що набір інваріантних заходів вихідного ланцюга є сукупністю всіх опуклих комбінацій π_i. Проте, якщо стан j аперіодичний, то $\lim_{n\to\infty} p_{jj}^{(n)} = \frac{C}{M_j}$, а для будь-якого іншого стану i, нехай f_{ij} є ймовірністю того, що ланцюг коли-небудь відвідає стан j, якщо він починається в i, то $\lim_{n\to\infty} p_{ij}^{(n)} = C\frac{f_{ij}}{M_j}$.

Якщо стан i є періодичним з періодом $k > 1$, то межа $\lim_{n\to\infty} p_{ii}^{(n)}$ не існує, хоча межа $\lim_{n\to\infty} p_{ii}^{(kn+r)}$ існує для кожного цілого r.

Ланцюг Маркова не обов’язково повинен бути однорідним за часом, щоб мати рівноважний розподіл. Якщо існує розподіл ймовірностей за станами π такий як (14) для кожного стану j і кожного n-го разу, тоді π є рівноважним розподілом ланцюга Маркова.

$$\pi_i = \textstyle\sum_{i\in S} \Pr(X_{n+1} = j | X_n = i). \tag{14}$$

Таке може виникнути у методах Монте-Карло ланцюга Маркова в ситуаціях, коли використовується ряд різних перехідних матриць, оскільки кожна є ефективною для певного виду змішування, але кожна матриця поважає спільний розподіл рівноваги.

Кажуть, що ланцюг Маркова є оборотним, якщо існує розподіл ймовірностей π над його станами таким, що для всіх разів n та всіх станів i та j. Ця умова відома як детальний стан балансу (деякі книги називають її місцевим рівнянням балансу).

$$\pi_i \Pr(X_{n+1} = j | X_n = i) = \pi_j \Pr(X_{n+1} = i | X_n = j). \quad (15)$$

Розглядаючи фіксований довільний час n та використовуючи скорочення $p_{ij} = \Pr(X_{n+1} = j | X_n = i)$, докладне рівняння балансу можна записати компактніше наступним чином:

$$\pi_i p_{ij} = \pi_j p_{ji}. \quad (16)$$

Критерій Колмогорова дає необхідну та достатню умову, щоб ланцюг Маркова був оборотним безпосередньо з імовірностей перехідної матриці. Критерій вимагає, щоб продукти ймовірностей навколо кожного замкнутого циклу були однаковими в обох напрямках навколо циклу.

Оборотні ланцюги Маркова поширені в підходах до ланцюга Маркова Монте-Карло, оскільки детальне рівняння балансу для бажаного розподілу π обов'язково означає, що ланцюг Маркова побудований так, що π – стаціонарний розподіл. Навіть з неоднорідними ланцюгами Маркова, де використовуються множинні перехідні матриці, якщо кожна така матриця переходу демонструє детальний баланс з потрібним π розподілом, це обов'язково означає, що π – стаціонарний розподіл ланцюга Маркова.

Дослідження повідомили про застосування та корисність ланцюгів Маркова в широкому діапазоні тем, таких як фізика, хімія, біологія, медицина, музика, теорія ігор та спорт.

Зокрема, Марківські системи широко з'являються в термодинаміці та статистичній механіці, коли ймовірності використовуються для представлення невідомих або немодельованих деталей системи, якщо можна припустити, що динаміка є інваріантною за часом, і що не потрібно розглядати відповідну історію, яка вже не включена в описі стану.

Реакційна мережа - це хімічна система, що включає безліч реакцій та хімічних видів. Найпростіші стохастичні моделі таких мереж трактують систему як безперервний часовий ланцюг Маркова, при цьому стан має кількість молекул кожного виду та реакції, змодельовані як можливі переходи ланцюга. Ланцюги Маркова і Марківські процеси безперервного часу корисні в хімії, коли фізичні системи близько наближаються до властивості Маркова. Наприклад, уявіть велику кількість n молекул в розчині в стані A, кожна з яких може пройти хімічну реакцію до стану B з певною середньою швидкістю.

Марківські ланцюги широко застосовуються також і в біології, зокрема, у філокінетиці та біоінформатиці, де більшість моделей еволюції ДНК використовують ланцюги Маркова безперервного часу для опису нуклеотиду, присутнього на даній ділянці геному; динаміці популяції, де ланцюг Маркова, зокрема, є центральним інструментом теоретичного вивчення матричних моделей популяції; нейробіології, де використовуються ланцюги Маркова, наприклад, для імітації неокортесу ссавців та системній біології, наприклад, з моделюванням вірусної інфекції поодиноких клітин.

Ланцюги Маркова використовуються в алгоритмічній музичній композиції, зокрема в таких програмних засобах, як Csound, Max та SuperCollider. У ланцюзі першого порядку стани системи стають значеннями ноти чи тону, будується вектор ймовірності для кожної ноти, заповнюючи матрицю ймовірностей переходу. Алгоритм побудований для отримання значень вихідних нот на основі зважування матриці переходу, які можуть бути значеннями примітки MIDI, частотою або будь-яким іншим бажаним показником.

Марківський ланцюг другого порядку може бути введений, враховуючи поточний стан, а також попередній стан. У вищі ланцюги n-го порядку, як правило,

"групують" окремі ноти, одночасно "відриваючись" від інших моделей та послідовностей. Ці ланцюги вищого порядку, як правило, дають результати з почуттям фразової структури, а не «безцільним блуканням», виробленим системою першого порядку.

Зазвичай музичним системам потрібно застосовувати специфічні умови контролю на обмежених породженими ними кінцевих послідовностях, але обмеження управління не сумісні з марківськими моделями, оскільки вони індукують далекі залежності, що порушують марковську гіпотезу обмеженої пам'яті. Для подолання цього обмеження запропоновано новий підхід.

Також приховані моделі Маркова є основою для більшості сучасних систем автоматичного розпізнавання мовлення; генераторах тексту [11]; PageRank веб-сторінки, які використовуються в Google також визначаються ланцюгом Маркова; у статистиці, моделюванні різних природніх або соціологічних процесів; а також у випадковій генерації в багатьох іграх.

Розподіл ймовірностей переходів зазвичай представляють у вигляді матриці. Якщо ланцюг Маркова має N можливих станів, то матриця матиме вигляд $N \times N$, у якій запис (I, J) буде ймовірністю переходу зі стану I у стан J. Побудуємо розподіл ймовірностей для поточної задачі.

Таблиця 1 – Розподіл ймовірностей для задачі генерування ландшафту

	Земля	Лід	Лава	Черв.Кам.	Криж.Кам.
Земля	0.5	0	0	0.25	0.25
Лід	0	0.75	0	0	0.25
Лава	0	0	0.75	0.25	0
Черв.Кам.	0.25	0	0.25	0.5	0
Криж.Кам.	0.25	0.25	0	0	0.5

Отож за допомогою ланцюгів Маркова задано множину усіх можливих станів (у нашому випадку – усіх можливих на майбутній локації видів ландшафту) та для

кожної з них задали множину ймовірностей переходів у усі можливі стани (включаючи самого себе), а також розмір ігрового світу чи рівня.

3.1.3 Випадкове генерування контенту ігрового рівня

Після цього розташуємо на локації контент, тобто ігрові об’єкти. Для цього слід задати мінімально можливу відстань між об’єктами, а також ймовірність виникнення кожного типу об’єкту на локації. Наприклад, у мілких монстрів шанс появи на локації буде значно вищим, аніж у якогось умовного легендарного дракона.

Потрібно також ретельно задати правила генерування для кожного з об’єктів, а також мінімальну та максимальну відстань від цього об’єкта до інших.

Нехай матимемо по два типи істот для генерації на кожному з типів ландшафту – одна з них матиме більш високий шанс з’явитись на локації, друга менший. А на ландшафті типу лави та криги додатково буде невеликий шанс на появу босів, тобто сильних монстрів.

Нехай також матимемо кілька типів пасток та кілька типів нагород, що також зможуть виникати із різними ймовірностями відповідно до типу ландшафту.

У бідь-якій грі необхідно ретельно продумати баланс сили героя та монстрів [12], та для автозгенерованого контенту це критичніше. На це можуть впливати не лише якісь власні характеристики героя, а й ігрове оточення. Наприклад, на ділянці ландшафту типу лава у героя можуть поступово зменшуватись одиниці здоров’я, а на ділянці ландшафту типу криги знижуватись спритність. Монстри, що з’являються на цих ділянках можуть у свою чергу не лише піднімати свої характеристики залежно від характеристик головного персонажа, а й здобувати чи мати за замовчуванням якісь бонуси на тій ділянці, де вони з’явились та навпаки якість неприємні дебафи. Наприклад, монстр, що з’явився на

льоду, буде швидко та без зайвої допомоги слабшати та гинути на ландшафті типу лава.

За задачі генерації необхідно також продумати мінімальну та максимальну відстані між ігровими об’єктами. Потрібно продумати чи буде вона однаковою для всіх об’єктів чи для різних об’єктів зона, де поряд не буде інших об’єктів, буде різною; чи буде мінімальна відстань до наступних об’єктів змінюватись залежно від того який об’єкт намагаються розмістити поряд із попереднім, тобто чи буде щільність виникнення, наприклад, мілких монстрів, менша ніж щільність виникнення великого монстра поряд із мілкими монстрами.

Після вирішення вищезазначений питань, генеруємо об’єкт за заданими правилами з’явлення та згідно заданим ймовірностям їх появи у випадковій проходимій точці ландшафту, такій, що не мала б у радіусі мінімально можливої відстані між об’єктами іншого об’єкта. Умова завершення роботи алгоритму досягається тоді, коли немає таких точок, які б задовольняли умову генерування нового об’єкту.

3.2 Розробка програмного забезпечення для генерування ландшафту

У рамках роботи було спроектовано реалізацію алгоритму генерування ландшафту на базі ланцюгів Маркова.

Програмний додаток для демонстрації роботи алгоритму генерування ландшафту 2D мапи для комп’ютерної гри спроектований на мові програмування C# із використанням технології Windows Forms. C# – це об'єктно-орієнтована мова програмування, що розроблена в 1998-2001 роках групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберг і Скотта Вільтаумота як мову розробки додатків для платформи Microsoft .NET Framework [13]. C# відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження

операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML. Перейнявши багато від своїх попередників - мов C++, Delphi, Модула, Smalltalk і, особливо, Java – C#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C#, на відміну від C++, не підтримує множинне успадкування класів (між тим допускається множинне спадкування інтерфейсів). C# розроблявся як мова програмування прикладного рівня для CLR (загальномовне виконавче середовище, тобто виконавча середа для байт-коду, у який компілюються програми, що написані на .NET-сумісних мовах програмування) і, як така, що залежить, перш за все від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих чи інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльованою в відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід очікувати і в подальшому (проте, ця закономірність була порушена з виходом C# 3.0, що представляє собою розширення мови, що не спираються на розширення платформи .NET). CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, прибирання сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так же, як це робиться для програм на VB.NET, J# і ін. Windows Forms – інтерфейс програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework [14]. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код – класи, що реалізують API для Windows Forms, що не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні ПЗ на C#, C++, так і на VB.Net, J# і ін. З одного боку, Windows Forms розглядається як заміна більш старої і складної бібліотеці MFC, спочатку написаної на мові C++. З

іншого боку, WF не пропонує парадигму, порівнянну з MVC. Для виправлення цієї ситуації і реалізації даної функціональності в WF існують сторонні бібліотеки.

Обрані мова та інтерфейс програмування додатків дозволяють спростити та значно прискорити розробку, а також надають можливість з переносу програми на інший інтерфейс користувача, адже мова програмування C# підтримується багатьма ігровими рушіями, зокрема доволі популярним рушієм – Unity. Unity – це кросплатформове середовище розробки комп'ютерних ігор, що розроблена американською компанією Unity Technologies [15]. Воно дозволяє створювати додатки, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші. Основними перевагами Unity є наявність візуальної середовища розробки, кросплатформової підтримки і модульної системи компонентів. На Unity написані тисячі ігор, додатків, візуалізації математичних моделей, які охоплюють безліч платформ і жанрів, цей рушій можуть вільно використовувати як великі ігрові розробники, так і незалежні студії.

Інтерфейс програми доволі мінімалістичний, має лише найнеобхідніші для демонстрації роботи алгоритму елементи інтерфейсу: поле для задання розміру мапи, поле для відображення мапи та кнопку генерування.

У програмі реалізовано описаний вище алгоритм генерування ігрового ландшафту для 2D мапи із квадратною формою комірок. Алгоритм базується на використанні ланцюгів Маркова, який здатен генерувати ігрову мапу заданого перед генерацією розміру. Ландшафт генерується поклітинно за порядком, а на генерування типу кожної клітинки впливають оточуючі її комірки, тобто для кожної комірки x_{ij} ймовірності типів ландшафтів вимірюються за формулою:

$$P(T)_{x_{(i,j)}} = P(T)_{x_{(i,j-1)}} * P(T)_{x_{(i-1,j)}} * P(T)_{x_{(i-1,j)}} * P(T)_{x_{(i-1,j+1)}}, \quad (15)$$

де T – тип ландшафту, x_{ij} – комірка, для якої потрібно згенерувати тип ландшафту, $P(T)_{x_{(i,j)}}$ – ймовірність виникнення типу ландшафту.

На рисунку 3.4 наведено графічний інтерфейс користувача програмного додатку для демонстрації роботи алгоритму.

Рис. 3.4 – Скріншот програмної реалізації алгоритму генерації ландшафту

Поточна демонстраційна реалізація передбачає зберігання згенерованої мапи у вигляді двовимірного масиву типів ландшафту, що значно спрощує задачу міграції коду на іншу графічну платформу.

Загалом результати роботи алгоритму генерації ландшафту ігрової локації задовільні, локації досить різноманітні, тож використання такого алгоритму може значно пришвидшити розробку ігор та сильно урізноманітніти їх.

Проте, поточний демонстраційний застосунок не передбачає повноцінної ігрової реалізації, тож тут немає таких важливих речей, як генерація монстрів та предметів, а також їх баланс, що також є не менш важливою частиною гри, адже без наявності власне ігрового застосунку перевірити баланс буде неможливо.

ВИСНОВКИ

У рамках виконання атестаційної роботи було виконано ретельний аналіз існуючих методів розробки віртуальних світів у відеоіграх. Було розглянуто прийоми як ручного, так і автоматичного генерування віртуальних світів.

Було ретельно розглянуто основні засади створення віртуальних світів у відеоіграх вручну, такі, як увага на просторі, використання прийомів візуальної мови, організація руху, заохочення дослідження ігрового оточення та інше.

Більш детально було розглянуло питання автоматичної генерації віртуальних світів, зокрема рівнів або локацій відеоігор. Розглянуто деякі існуючі алгоритми генерування віртуальних світів на прикладі існуючих та популярних ігор. Зокрема приклад гарної продуманої, цікавої та доречної випадкової генерації віртуального світу на прикладі гри Sid Meier's Civilization V. Також було розглянуто приклад невдалої реалізації випадкової генерації віртуальних світів на прикладі гри No Men’s Sky.

Було спроектовано алгоритм автоматичної генерації віртуального світу за заданими параметрами ландшафту та із заданими параметрами генерування інтерактивних ігрових об’єктів на основі ланцюгів Маркова та заданих ймовірностей виникнення заданих типів ландшафту, а також заданих типів ігрових об’єктів (разом із ймовірностями їх виникнення).

ПЕРЕЛІК ДЖЕРЕЛ

1. Fencot C., Clay J., Lockyer M., Massey P. Game Invaders: The Theory and Understanding of Computer Games — Hoboken: Wiley, 2012. — 230 с.

2. Muehl W., Novak J. Game Development Essentials: Game Simulation Development — Boston: Cengage Learning, 2007. — 272 с.

3. Ebert D., Musgrave F., Peachey D., Perlin K., Worley S. Texturing and Modeling: A Procedural Approach — Burlington: Morgan Kaufmann, 2002. — 687 с.

4. Кадиков М. Проектирование виртуальных миров. Теория и практика дизайна уровней — Москва: Издательские решения, 2019. — 398 с.

5. Kenwood J., Gain J., Marais P. Efficient Procedural Generation of Forests // Journal of WSCG. 2014. Vol.22, №1. С. 31 — 38.

6. Génevaux J.-D., Galin E., Guérin E., Peytavie A., Benes B. Terrain Generation Using Procedural Models Based on Hydrology // ACM Trans. Graph. 2013. Vol. 32, № 4. С. 143:1 — 143:13.

7. Zhou A., Qu B.-Y., Li H., Zhao S.-Z., Nagaratnam P., Zhang Q. Multiobjective evolutionary algorithms: A survey of the state of the art // Swarm and Evolutionary Computation. 2011. Vol. 1, № 1. С. 32 — 49.

8. Smith G. Expressive Design Tools: Procedural Content Generation For Game Designers // Ph.D. dissertation, University of California, 2012. URL: http://escholarship.org/uc/item/0fn558gq#page-4 (дата звернення: 13.03.2020)

9. Cannizzo A., Ramírez E. Towards Procedural Map and Character Generation for the MOBA Game Genre // Ingeniería y Ciencia. 2015. Vol.11, №12. С. 12 — 13.

10. No Man’s Sky: Обзор // StopGame.Ru. URL: https://stopgame.ru/show/84261/no_man_s_sky_review (дата звернення: 21.03.2020)

11. Тред: Нелепые сгенерированные «пацанские» цитаты // TJournal. URL: https://tjournal.ru/internet/100591-tred-nelepye-sgenerirovannye-pacanskie-citaty (дата звернення: 21.03.2020)

12. Новіков Ю.С., Давидов О.П. Оптимізація ігрового балансування стратегій з використанням математичних алгоритмів теорії ігор // 22-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6. — Харків: ХНУРЕ. 2019. — 496 с.

13. Либерти Д. Программирование на C#. — Санкт-Петербург: Символ-Плюс, 2003. — 688 с.

14. Sells C. Windows Forms Programming in C#. — Boston: AddisonWesley Professional, 2003. 682 с.

15. Бонд Дж.-Г. Unity и C#. Геймдев от идеи до реализации: 2-е изд. — Санкт-Петербург: Питер Пресс, 2019. — 928 с.