МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Програмування
(повна назва кафедри)

Магістерська робота

РОЗРОБКА ГРИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ РЕЙКАСТИНГУ НА ОСНОВІ ВЛАСНОГО РУШІЯ ДЛЯ ПСЕВДО-3D ГРАФІКИ

Виконав: студент групи ПМіМ-21с,
спеціальності
122 - «Комп'ютерні науки»
(шифр і назва спеціальності)

(підпис) Ковбасюк О.В.
(прізвище та ініціали)

Керівник (підпис) Заболоцький Т.М.
(прізвище та ініціали)

Рецензент (підпис) Мельничин А.В.
(прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Програмування
Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри доц. Ярошко С.А.
"13" вересня 2021 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Ковбасюку Остапу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *РОЗРОБКА ГРИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ РЕЙКАСТИНГУ НА ОСНОВІ ВЛАСНОГО РУШІЯ ДЛЯ ПСЕВДО-3D ГРАФІКИ*

керівник роботи Заболоцький Тарас Миколайович, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **"13" вересня 2022 року № 15**

2. Строк подання студентом роботи **08.12.2022р .**

3. Вихідні дані до роботи *інтернет-ресурси за темитикою роботи .*

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)

а) створити власний рушій для псевдо-3D графіки
б) скласти специфікацію вимог до програмного продукту
в) розробити прототипи майбутнього користувацького інтерфейсу гри
г) розробити гру, з використанням раніше створених прототипів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

а) ілюстрації, що демонструють роботу алгоритмів
б) ілюстрації, що відображають прототипи користувацького інтерфейсу
в) ілюстрації, що зображають отримані результати та ігровий процес

6. Дата видачі завдання **1 вересня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Постановка завдання	*вересень*	*виконано*
2.	Створення специфікації вимог до розроблюваного ПЗ	*вересень*	*виконано*
3.	Реалізація рушія для псевдо-3D графіки	*жовтень*	*виконано*
4.	Розробка прототипів користувацького інтерфейсу	*жовтень*	*виконано*
5.	Розробка користувацького інтерфейсу	*жовтень*	*виконано*
6.	Модифікація рушія для відображення статичних зображень	*жовтень*	*виконано*
7.	Оформлення магістерської роботи	*листопад*	*виконано*
8.	Підготовка до захисту магістерської роботи	*грудень*	*виконано*

Студент _____ **Ковбасюк О.В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Заболоцький Т.М.**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська робота:

Мета роботи – розробка веб гри за допомогою власного рушія для генерації псевдо-3D графіки розробленого на основі технології рейкастингу.

Перший розділ містить інформацію про мету створення програмного забезпечення, описується предметна область.

Другий розділ описує основні характеристики системи, якими вона має володіти, а також містить чітку специфікацію вимог до розроблюваного програмного продукту.

Третій розділ містить інформацію про процес прототипування майбутньої системи та описує основні шляхи взаємодії з системою за допомогою даних прототипів.

Четвертий розділ описує інформацію, що стосується самої розробки програмного продукту - основні класи системи, їх методи та властивості, описані найважливіші частини коду, представлені результати розробки.

Ключові слова: рейкастинг, графіка, лабіринт, інтерфейс, псевдо-3D, програмний продукт.

ЗМІСТ

ВСТУП

Комп’ютерні ігри сьогодні складають не найменшу частину світової галузі. Майже кожен пробував, а ті що й ні, все одно знайомі з ними. Зазвичай в кожній підмножині людей, завжди є той, хто постійно грає у ігри. Але це не дивує, адже відеоігри придумали не один десяток років тому, і з того часу, вони почали охоплювати все більшу кількість ігрових платформ, стартуючи аркадними системами чи домашніми консолями і закінчуючи мобільними пристроями та персональними комп’ютерами. А розпочали вони свій шлях у лабораторіях науковців.

Наприклад, перші “хрестики-нулики”, а точніше “OXO” у 1952 році створив британський професор А.С. Дуглас в Кембриджському університеті як частину своєї дисертації.

У 1962 році було винайдено Spacewar! Стівом Расселом. Це була космічна комп'ютерна гра для найпотужніших тоді комп'ютерів, які в основному знаходилися тільки в лабораторіях університетів. Однак, на той момент відеогра була доволі примітивою по мірках сьогодення.

Невдовзі, у 1969 році Артур Аппель представив перший в своєму роді алгоритм рейкастингу[1], який використовувався для промальовування 3D об’єктів.

Wolfenstein 3D – це назва комп’ютерної гри, яка 1992 року, стала першою грою, розробленою на основі технології кидання променів і зазнала неабиякого успіху. Інтерактивний 3D світ, відображення з вищою частотою кадрів і рівнем фотореалізму, ніж у всіх попередніх ігор – саме це привело до успіху та змінило індустрію відеоігор.

Метою виконання даної магістерської роботи розробка веб-додатку у формі гри, на базі власного рушія, який демонструє технологію рейкастингу для відображення псевдо-3D графіки.

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВСТАНОВЕЛННЯ МЕТИ ВИКОНАННЯ РОБОТИ

1.1. Огляд різновидів жанрів комп'ютерних ігор.

Сьогодення важко уявити без комп'ютерних ігор. Це одне з найпоширеніших занять, адже з їхньою допомогою можна, зануритися в інший світ, пережити моменти, які навряд чи коли-небуть стануться в реальності. Це дуже схоже на кіно, але ми можемо взаємодіяти з навколишнім середовищем та персонажами, що населяють цей світ. Десь ваш вибір напряму впливає на сюжет і усі подальші події, десь людина може спробувати себе в ролі пілота, стратега, будівельника чи навіть хірурга, може керувати арміями, цивілізаціями, тощо. Існує багато видів ігор і їх жанрів, тому варто виділити основні з них.

Екшн. Один з найпоширеніших та найбільш базових жанрів. Тут від гравця вимагається максимальна увага, гарні рефлекси та хороша реакція аби подолати ворогів та інші обставини. Ці ігри зазвичай пов'язані з подіями, які швидко змінюються, як от перестрілки, погоні, тощо[2]. Даний жанр ділиться на такі основні піджанри:

- **Шутери** – ігри, що відображають боротьбу з ворогами засобами стрільби. Зазвичай вони поділяються на ігри від **першого** чи **третього** лиця відповідно до перспективи камери.
- **Файтинги** – ігри, що імітують техніки ближнього бою, в основному це бої 1 на 1 на різноманітних бойових аренах.
- **Beat 'em up** – схожі на файтинги, відрізняються одночасно неймовірною кількістю одночасних появ ворогів та гадають можливість вільно ходити по світу, а не в межах конкретної арени.
- **Платформери** – основна задача гравця доставити персонажа із точки А в точку Б по платформах з попутним нищенням ворогів та подоланням перешкод.
- **Лабіринти –** це ігри головоломки, де ціль гравця - знайти вихід, збирати необхідні для виходу предмети, має уникати пастки на шляху до виходу.

Стратегія. В цих іграх планування дій, тактик та стратегій є основною задачею для досягення певної цілі, наприклад, загарбання якоїсь території, створення цивілізації, розвиток замку, тощо. В один момент часу гравець керує не лише своїм персонажем, а, до прикладу, армією, містом, цивілізацією. Тут масштаб часу дуже сильно відрізняється від реального, 1 ігрова година це декілька хвилин або ж секунд реального часу, відповідно якесь будівництво чи тренування піхотинця теж триває кілька хвилин або секунд[3]. Згідно з організацією ігрового часу, є два основні різновиди стратегій:

- **Покрокові стратегії** – стратегії, де гравець та супротивник роблять певні дії покроково, по черзі, і можуть за один хід виконати лише деяку обмежену множину операцій.
- **Стратегії в режимі реального часу** – обидві сторони діють одночасно. Вони ще поділяються на **Tower Defense –** де аби не пропустити ворогів, гравець будує оборонні вежамі, а вороги рухаються до деякої цілі хвилями. Та **MOBA (Multiplayer Online Battle Arena) –** тут гравець керує своїм персонажем і є частиною команди онлайн гравців, разом вони захищають свою базу та мають за мету знищити ворожу.

Рольова гра. Тут гравець діє відповідно до правил ролі свого персонажа, наприклад, маг не може робити певні дії, притаманні лицареві чи лицар не може виконувати дії що і лучник, тощо. Основна мета гри – виконання завдань, підвищення рівня гравця, покращенню спорядження для досягнення її максимальної якості. Особливим видом рольової гри є **MMORPG (Massively multiplayer online role-playing game) –** це багатокористувацькі онлайн рольові ігри, де гравці можуть взаємодіяти одне з одним через інтернет[4].

Симулятор. Ігри даного жанру симулюють певні обставини, навколишнє середовище, інструментарій, дуже схоже до реального світу. Є різні піджанри, такі як технічні (управління всякими технічними пристроями, авіацією, воєнною технікою, і тп.), аркадні (на відміну від аркад мають спрощену фізичну модель), економічні, спортивні, та інші[5].

Пригоди. В даних відеоіграх головний героєм йде по певній сюжетній лінії, яка може розгалужуватись, виконує якісь завдання, що відкривають можливість переходу на інший етап гри та можливість продовження історії. Це супроводжується розгадуванням загадок і потребує неабиякої уважності та логіки. В них сюжет розвивається доволі динамічно і є насичений багатою кількістю яскравих подій, різкими змінами обстановки, а персонажі, в свою чергу, проявляють кмітливість та сміливість, а не грубу силу[6]. Основні піджанри:

- **Інтерактивна література** – сюжет даних ігор не є жорстко фіксованим і може змінюватися відповідно до вибору гравця.
- **Інтерактивні фільми** – тут це міні фільми чи мультфільми, з'єднані між собою певною послідовністю, що постійно залежить від вибору гравця.

Варто згадати інші жанри:

- **Настільна гра** – настільні ігри в програмній реалізації .
- **Головоломки –** ігри, що складаються з вирішення завдань на логіку, передбачення якихось ситуацій та наслідків виконаних дій.
- **Games with a Purpose –** з їх допомогою гравці можуть використовувати знання, набуті в деякій сфері діяльності, для вивчення і засвоєння нової корисної інформації під час ігрового процесу.

1.2. Ray casting – технологія генерації псевдо 3D графіки

Ігри сьогодення це надзвичайні витвори мистецтва, в яких є неймовірна графіка, чудові сюжети та захоплюючі ігрові механіки. А за допомогою віртуальної реальності можливість заглибитися у віртуальний світ надає надзвичайно більше можливостей для потенційних гравців.

Рис.1.1. Скриншот гри Call of Duty: Modern Warfare II 2022

Але, звісно, так було не завжди. Саме за допомогою псевдо 3D технологій були створені перші “тривимірні” ігри. За допомогою деяких калькуляцій звичайні двовимірні елементи перетворювалися на деякі 3D проекції. Саме такою технологією і є рейкастинг.

Рейкастинг чи кидання променів (ray casting) – технологія для промальовування 3D зображення, що базується на методі зіткнення променів з поверхнею. Це один з найпростіших і найшвидших алгоритмів для творення комп’ютерної графіки, саме тому він став одним з найефективніших методів в ранніх 3D відеоіграх. Ця технологія займає надзвичайно малі ресурси комп’ютера, адже кидання променя є досить простим і не потребує багато математичних операцій і обчислень.

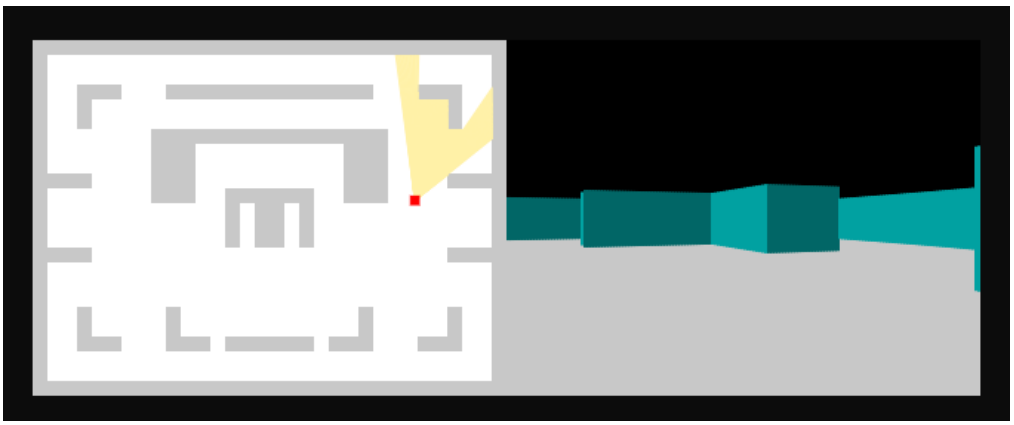

Рис.1.2. Приклад результату кидання променів.

РОЗДІЛ 2. ХАРАКТЕРИСТИКИ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

2.1. Характеристики продукту

Розроблюваний програмний продукт має володіти такими можливостями:

- Реєстрація та авторизація користувача в системі
- Гра без авторизації (режим гостя);
- Вільне пересуваня по рівню;
- Відображення мінікарти з позицією гравця на ній;
- Можливість збереження гри на будь-якому етапі, та її продовження (користувач зареєстрований);
- Генерація рівнів-лабіринтів випадковим чином;
- Відображення часу проходження рівнів;
- Формування таблиці лідерів;
- Вибір варіанту текстурування рівня;
- Збір предметів для відкриття виходу з рівня.

2.2. Специфікація вимог

2.2.1. Реєстрація користувача

Опис і пріоритет

На головній сторінці гри має бути кнопка для реєстрації. Після натиску на неї, користувач потрапляє на сторінку реєстрації. Ввівши необхідні дані та підтвердження, користувач повинен бути успішно зареєстрованим у системі.

Пріоритет: середній. Послідовність дія/відгук

Перейшовши на сайт та зареєструвавшись, користувач зможе використовувати функції, що наявні для саме зареєстрованих користувачів.

Функціональні вимоги

REQ-1.1: Має відбуватися перевірка введення даних та відображатися помилка при їх неправильному вводі.

REQ-1.2: Кнопка для завершення реєстрації повинна бути активною лише з правильно заповненими обов’язковими полями.

REQ-1.3: Якщо користувач з таким самим логіном існує в ситемі, має виводитиись відповідне повідомлення.

2.2.2. Авторизація в системі

Опис і пріоритет

На веб-сторінці гри, користувача зустрічає форма авторизації. Ввівши раніше зареєстровані дані для входу та підтвердивши форму, користувач успішно заходить у систему.

Пріоритет: середній.

Послідовність дія/відгук

Після навігації на сайт, користувач входить в систему і надалі може користуватись функціоналом, наявним лише авторизованим користувачам.

Функціональні вимоги

REQ-2.1: Кнопка для підтвердження входу має бути активною лише у випадку правильного набору даних.

REQ-2.2: Має відбуватися перевірка введення, якщо введені дані не правильні - відображення відповідної помилки.

REQ-2.3: Якщо користувач ввів не існуючий в системі логін, чи пароль до відповідного логіну не підходить - відображення помилки.

2.2.3. Можливість гри як гість (без авторизації)

Опис і пріоритет

На веб-сторінці гри, користувач бачить кнопку для гри як гість, натиснувши на яку, він має можливість грати без додаткового функціоналу (збереження і відновлення ігрового процесу, перегляд таблиці лідерів).

Пріоритет: високий.

Послідовність дія/відгук

Потрапивши на сайт, гравець натискає кнопку “Грати як гість” та може розпочати ігровий процес.

Функціональні вимоги

REQ-3.1: Кнопка гри в режимі гостя має бути завжди наявною для

неавторизованого користувача.

2.2.4. Можливість вільної навігації по карті

Опис і пріоритет

Розпочавши гру, користувач має бути доступна можливість вільної навігації по ігровому рівню.

Пріоритет: високий.

Послідовність дія/відгук

За допомогою натиску на кнопки: стрілка вгору, вниз, вправо та вліво при грі з комп'ютера, або ж на наекранні кнопки у випадку гри з мобільних пристроїв, користувач має переміщуватись по рівню.

Функціональні вимоги

REQ-4.1: Гравцеві має бути представлена можливість для повороту камеру ліворуч та праворуч а також можливість пересуватись вперед і назад за допомогою відповідних кнопок.

REQ-4.2: Гравець не має проходити через стіни (наявність розпізнавання колізії).

2.2.5. Перегляд таблиці лідерів

Опис і пріоритет

Після успішного закінчення рівня повинна вдображатися таблиця лідерів та їх час проходження.

Пріоритет: середній. Послідовність дія/відгук

Пройшовши рівень на екран виводиться лідерна таблиця

Функціональні вимоги

REQ-5.1: Користувач має успішно завершити рівень.

REQ-5.2: Має відобразитись таблиця з результатами інших гравців.

2.2.6. Вибір текстурування рівня

Опис і пріоритет

Перед генерацією рівня повинна бути можливість для задавання використовуваної текстури.

Пріоритет: середній. Послідовність дія/відгук

На екрані з параметрами рівня, є опція вибору майбутньої текстури.

Вибравши необхідну текстуру та запустивши генерацію, рівень буде стилізований відповідно до обраної текстури.

Функціональні вимоги

REQ-6.1: Користувач обирає певну текстуру.

REQ-6.2: Генерується рівень з відповідною текстурою.

2.2.7. Відображення на мінікарті

Опис і пріоритет

Під час гри, положення ігрового персонажа буде відображатися на мінікарті.

Пріоритет: середній.

Послідовність дія/відгук

В лівому верхньому куті ігрового полотна буде промальовуватись міні-карта, на якій буде відображено поточне положення гравця.

Функціональні вимоги

REQ-7.1: Згідно з переміщеннями гравця, його положення на мінікарті також повинне мінятись.

2.2.8. Можливість збереження прогресу гри, та його відновлення

Опис і пріоритет

Під час гри, авторизований корисгравець матиме можливість зберегти та за потреби відновити поточний стан гри, обравши відповідну опцію в меню.

Пріоритет: середній.

Послідовність дія/відгук

Перейшовши в меню, появляться кнопки серед яких будуть “зберегти гру” та “відновити гру”. Обравши опцію збереження, даний момент гри зберігається в окремий запис з часовою міткою. Обравши відновлення процесу гри, появиться список з записами про збереження, після вибору потрібного запису, гра відповідного запису відновлюється.

Функціональні вимоги

REQ-8.1: При збережені, записується положення гравця на рівні, час проходження,

маска мінікарти.

REQ-8.2: При відновленні, положення гравця на рівні, час проходження, маска мінікарти відновлюються згідно з обраним записом.

2.2.9. Випадкова генерація рівнів

Опис і пріоритет

Запустивши рівень він буде випадково згенерований.

Пріоритет: середній.

Послідовність дія/відгук

Після вибору розмірності рівня на екрані його параметрів та підтвердження створення буде випадково згенерований рівень-лабіринт обраного розміру.

Функціональні вимоги

REQ-9.1: Повинен випадковим чином генеруватись рівень обраного розміру, створений лабіринт має мати лише один вихід.

2.2.10. Відображення часу проходження рівня

Опис і пріоритет

Після початку проходження рівня замірюється час проходження.

Пріоритет: низький.

Послідовність дія/відгук

Після успішного завершення рівня, користувач бачить за скільки часу він його пройшов.

Функціональні вимоги

REQ-10.1: На початку проходження рівня запускається секундомір

REQ-10.2: По завершенню проходження секундомір зупиняється та відображається його час.

2.2.11. Збір предметів (ключів)

Опис і пріоритет

Для відкриття виходу з рівня-лабіринту користувач має зібрати певні предмети-ключі.

Пріоритет: середній.

Послідовність дія/відгук

Під час генерації рівня на ньому будуть розміщені ключі, зібравши які, для користувача відкриється вихід з рівня-лабіринту.

Функціональні вимоги

REQ-11.1: При створенні рівня генеруються предмети-ключі

REQ-11.2: Коли предмет перебуває в певному радіусі від користувача, він збирається

REQ-11.2: Вихід з лабіринту відкривається після збору усіх предметів.

РОЗДІЛ 3. ПРОЕКТУВАННЯ РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ПРОДУКТУ

3.1. Архітектура системи

Система буде розроблятися на основі клієнт-серверної архітектури, таким чином можна відділити бізнес логіку зі збереження даних та відображення користувацької інформації.

Для забезпечення якості і підтримуваності коду будуть застосовані методології SOLID[7] та DRY[8]. Фронт-енд частина системи буде написана за допомогою Angular[9], таким чином можна забезпечити легке створення компонент, які можна буде перевикористовувати, сервіси для доступу до сервера будуть виконані на базі шаблону Singleton[10]. Аби користувацький інтерфейс не зависав при здійсненні запитів до сервера вони будуть здійснюватися асинхронно за допомогою RxJS[11].

Серверна частина функціонуватиме в середовищі NodeJS[12] і використовуватиме драйвер mongodb[13] для зв’язку з базою даних.

3.2. Проектування поведінки системи

Відповідно до функціональних вимог в системі буде 2 типи користувачів: зареєстрований та гість. Зареєстрований користувач окрім звичайного проходження рівнів також матиме можливість перегляду таблиці лідерів, збереження та відновлення стану гри. Кожен новий рівень буде згенерований абсолютно випадково у формі лабіринту. В даному лабіринті завжди можна буде попасти з однієї точки в іншу. Також, на рівні мають бути розставлені ключі, збір яких буде обов’язковим для відкриття виходу з лабіринту.

Сама ж генерація лабіринту буде здійснена за допомогою модифікації алгоритму бектрекінгу, так як в нашому випадку потрібно генерувати лабіринт, який має певну товщину стін. Нижче відображено блок-схему генерації рівня-лабіринту.

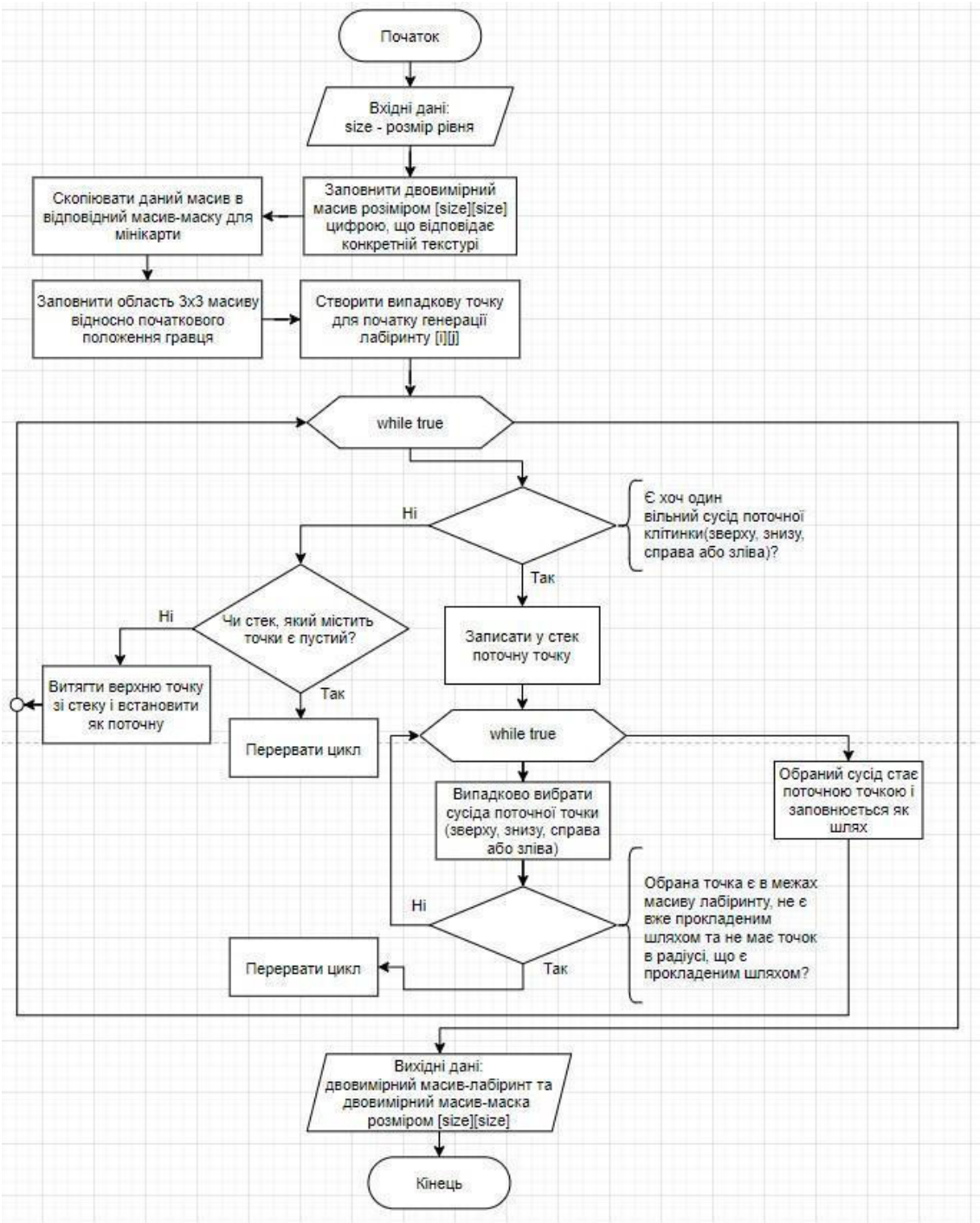

Рис.3.1. Блок-схема генерації карти-лабіринту

3.3. Прототипування інтерфейсу

За допомогою графічного редактора Figma, відповідно до функціональних вимог було створено прототипи графічного інтерфейсу програмного продукту. Прототипи створено в двох варіаціях - для комп’ютера а також для мобільного пристрою.

На головній сторінці гри користувача зустрічає форма авторизації, за

допомогою якої можна здійснити вхід, з використанням раніше зареєстрованих даних, або ж розпочати гру як гість.

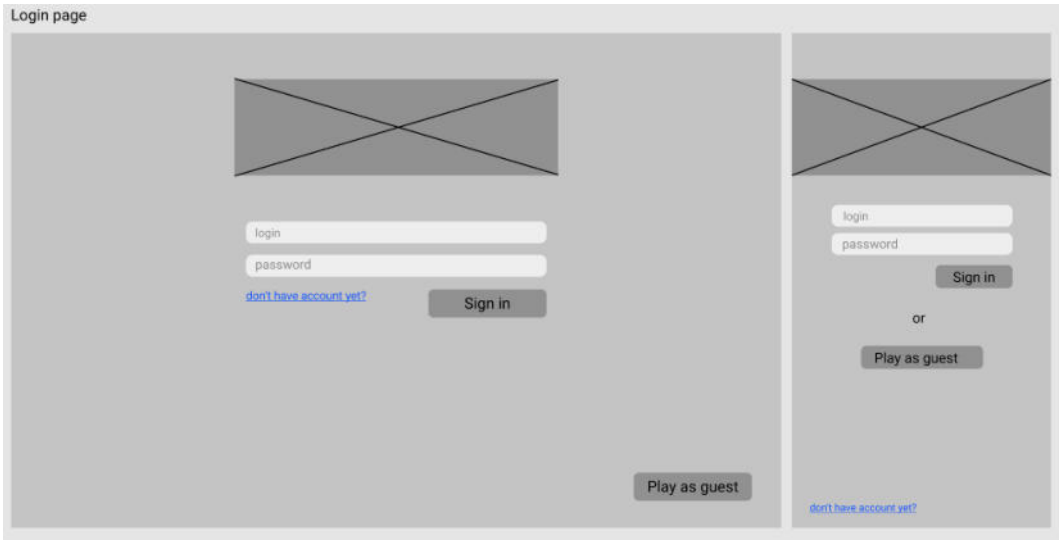

Рис.3.2. Прототип сторінки входу

Якщо користувач не створював акаунту раніше, він може натиснути на відповідну кнопку, тоді його перенаправить на форму реєстрації. Після успішного вводу даних та підтвердження, користувача буде зареєстровано і він зможе продовжити гру з використанням свого акаунту.

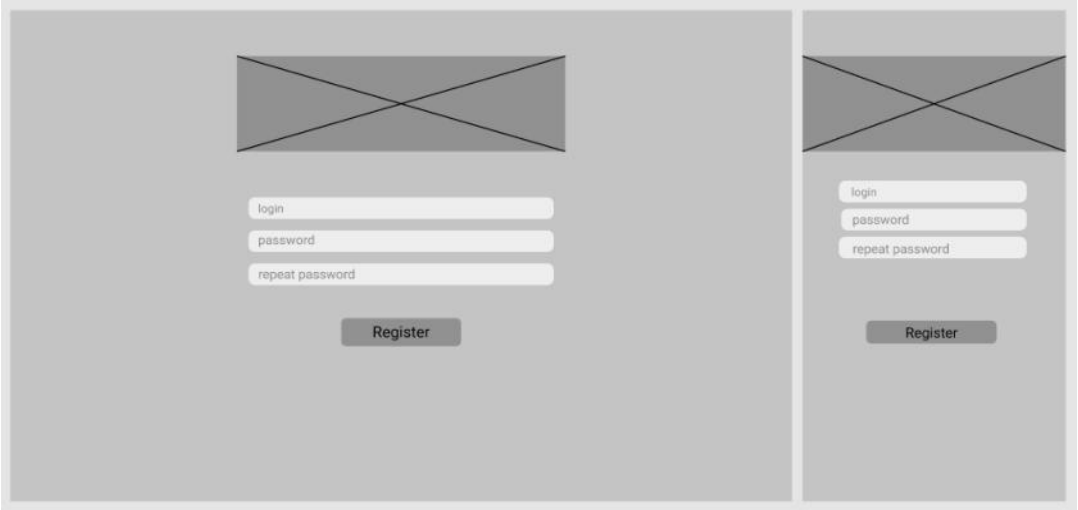

Рис.3.3. Прототип сторінки реєстрації

Після авторизації, користувача перенаправляє на сторінку з ігровим меню/чи одразу до вибору параметрів нового рівня (в залежності від типу користувача - гість чи зареєстрований).

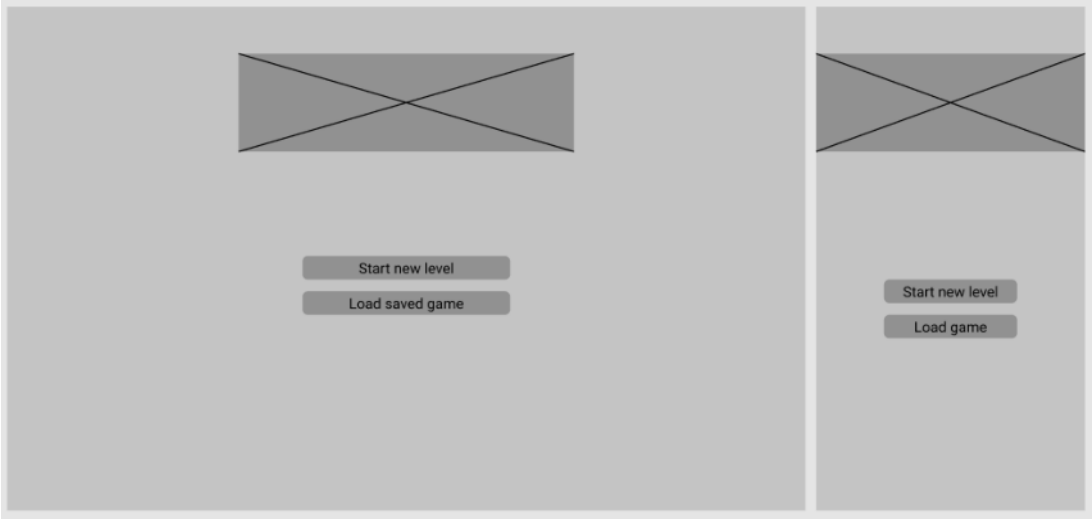

Рис.3.4. Прототип меню (для зареєстрованого користувача)

Далі, вибравши параметри рівня, чи завантаживши збереження, йде перенаправлення на саму сторінку гри з ігровим вікном, в якому є мінікарта, сама проекція рівня, наекранні кнопки (тільки на мобільному пристрої), тощо. Для гри з мобільного пристрою потрібно перейти в пейзажний режим.

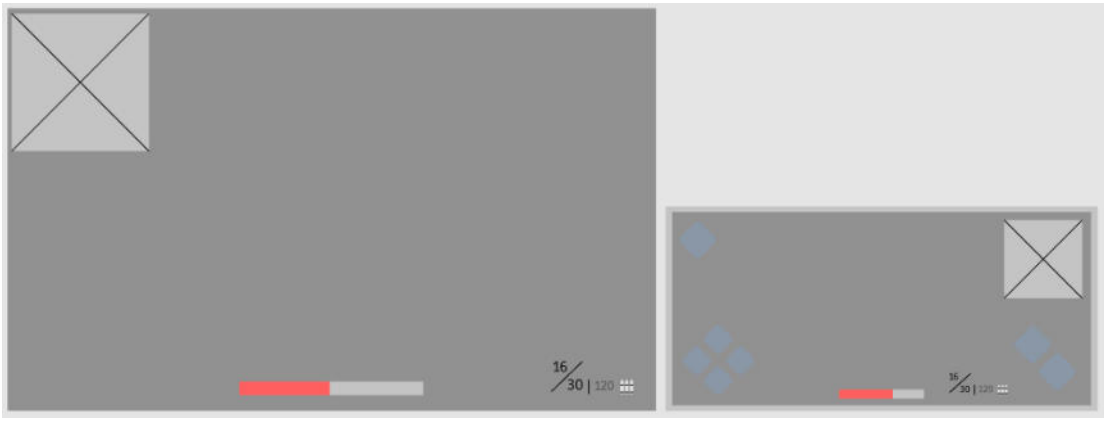

Рис.3.5. Прототип ігрового полотна

Перейшовши в загальне меню гри, користувач зможе здійснювати наступні дії: повернутися до гри, зберегтися чи відновити збережений стан гри

(зареєстрований користувач) чи вийти з гри

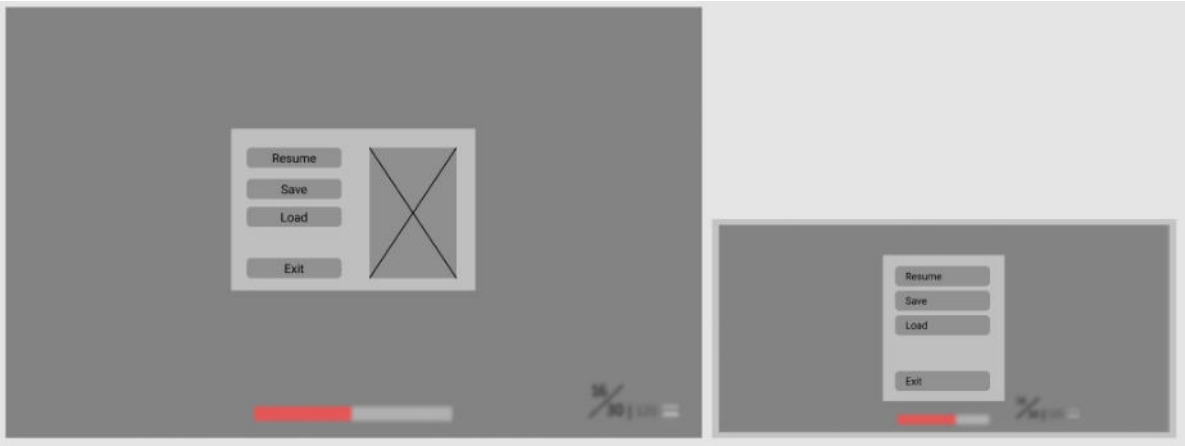

Рис.3.6. Прототип ігрового меню

Останнім прототипом є меню вибору збереженого стану гри для його відновлення. Там буде зображено записи, з часовими штампами. Після вибору одного із них поточний рівень буде змінений на відповідний із збереження.

Рис.3.7. Прототип меню вибору збереженого стану гри

РОЗДІЛ 4. МЕТОД КИДАННЯ ПРОМЕНІВ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

4.1. Метод кидання променів[14]

Механізм кидання променю і утворення проекції реалізовано наступним чином. З якоїсь початкової точки кидається промінь. Для того аби не шукати через кожну одиницю можливе зіткнення, вводиться інкремент, який визначається з певним кроком по X та У. Цей інкремент визначається для X та У окремо і залежить від розміру самої клітинки та кута кидання променю. Таким чином, поступово інкрементуючи точку ми знайдемо відстань до першого вертикального та/або горизонтального перетинів. Це можна легко побачити випустивши з однієї точки два випадкові промені.

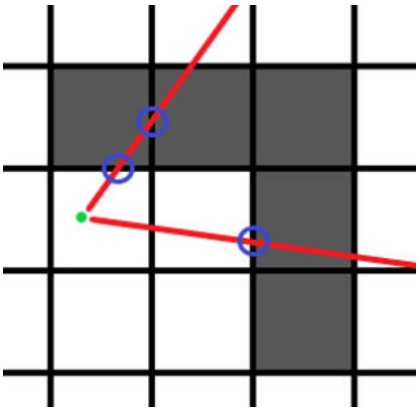

Рис.4.1. Результат випускання двох променів з однієї точки

Серед обох перетинів одного променю вибирається те, що є ближчим до його початку. Таким чином ми отримуємо реальну точку зіткнення зі стіною. Далі вираховуємо висоту стіни, згідно з отриманою відстанню променя та початковим заданим співвідношенням довжини променя до висоти: **(початкова висота стіни / відстань до стіни) * початкова відстань до стіни**. Дана висота відповідає тільки одному променю, тому виводом на екран буде один вертикальний стовпець вирахуваної висоти. Далі його можна заповнити певним кольором, або ж текстурою. Для цілісності картинки потрібно випустити певну кількість таких променів з деяким градусним кроком. Таким чином, випустивши до прикладу 64

промені з кроком градусу 1, ми отримуємо стільки ж вертикальних стовпців, після заповнення кольором та масштабування яких, вже вимальовується повноцінна проекція стін на карті.

Рис.4.2. Приклад утвореного зображення

4.2. Реалізація веб-застосунку

4.2.1. Опис основних класів системи

Основна логіка гри містить такі 3 складові: рівень, промінь та гравець.

Клас рівня (Level) відповідає за генерацію, задання точки виходу та накладанням маски на карті. Володіє такими властивостями:

- ctx – це контекст ігрового полотна, на якому малюється рівень;
- canvasWidth, canvasHeight – ширина і висота ігрового полотна;
- levelArray – двовимірний масив, вміст якого - номери текстур (0 – пуста клітинка, 1+ - номери текстур);
- mask – двовимірний масив маски рівня, що поступово очищується;
- tilesCount – розмірність одного виміру масива (масив 50x50, розмір виміру – 50);
- startingPoint – початкова точка генерації, що вибирається випадково;
- finishPoint – кінцева точка-вихід з лабіринту.

Та наступні методи:

- generateLevel – відповідає за випадкову генерацію рівня, побудований на основі алгоритму бектрекінгу та використовує такі методи класу:

isAvailableAnyNeighbour (чи є вільний сусід) та doesAnyRadiusNeighbourIsPartOfPath (чи сусід в радіусі не є частиною шляху). Після успішної генерації викликає метод clearMapBlock для очистки території для початкової та кінцевої точки лабіринту;

- isAvailableAnyNeighbour – перевіряє наявність в клітинки невідвіданого сусіда (зверху, справа, знизу, зліва)
- doesAnyRadiusNeighbourIsPartOfPath – перевіряє, чи сусідні клітинки не є частиною прокладеного шляху;
- clearMapBlock – очищає кліки в заданому радіусі певної точки;
- clearMaskTile – відкриває клітинку маски;
- isCollision – перевіряє колізію в певній клітинці (наявність стіни);
- draw – малює рівень та накладає маску;
- drawFinishPoint – малює кінцеву точку лабіринта;
- getLevelData – метод, що повертає детальну інформацію рівня, розмірність, сам масив рівня, маску, вихід. Застосовується для збереження та відновлення стану гри.

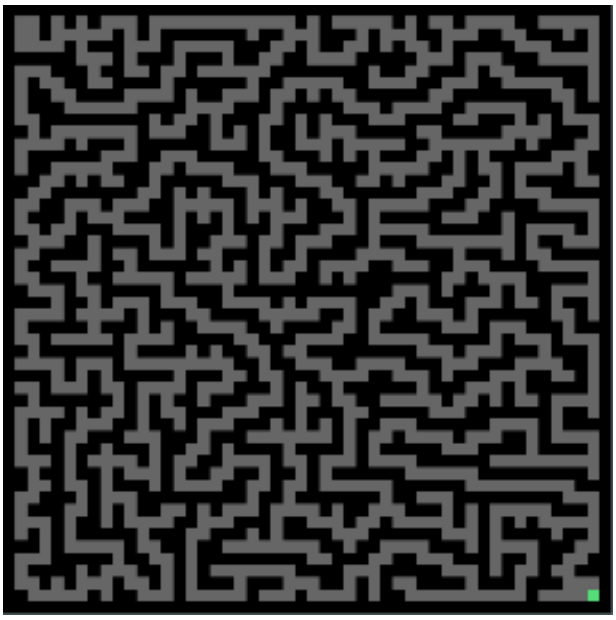

Рис.4.3. Результат створення рівня з відсутньою маскою

Клас променю (Ray) містить реалізацію алгоритму кидання променя та рендеру стін, на основі результатів обчислень. Основні властивості:

- ctx – контекст полотна, на якому буде промальовано промінь;
- level – посилання на створений рівень;
- x, y – координати початку променя;
- angle – глобальний кут променя на полотні;
- playerAngle - глобальний кут повороту гравця на полотні;
- angleIncrement – відхиллення від кута гравця;
- columnIndex – порядковий індекс променя;
- wallHitX, wallHitY – координати точки зіткнення променя зі стіною;
- distanceToWall –довжина променя.

Та основні методи:

- setAngleAndPosiiton – метод для оновлення позиції та кута повороту променя;
- cast – метод, що реалізовує алгоритм рейкастингу, результатами виконання якого є обчислена координата зіткнення зі стіною, відстань до стійни та ідентифікатор пікселя текстури, який буде використано при промальовуванні стіни в 3D проекції;
- draw – метод, що відмальовує промінь на карті;
- renderWall – метод, що проектує випущений промінь в псевдо 3D простір.

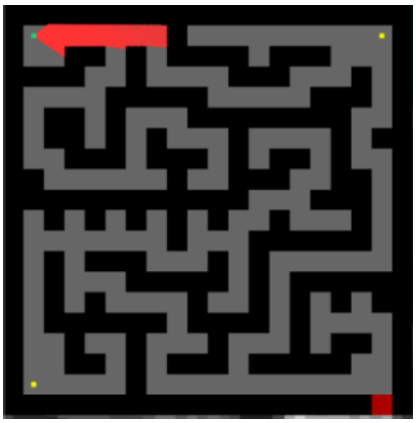

Рис.4.4. Карта з зображеними яскраво червоним кольором променями

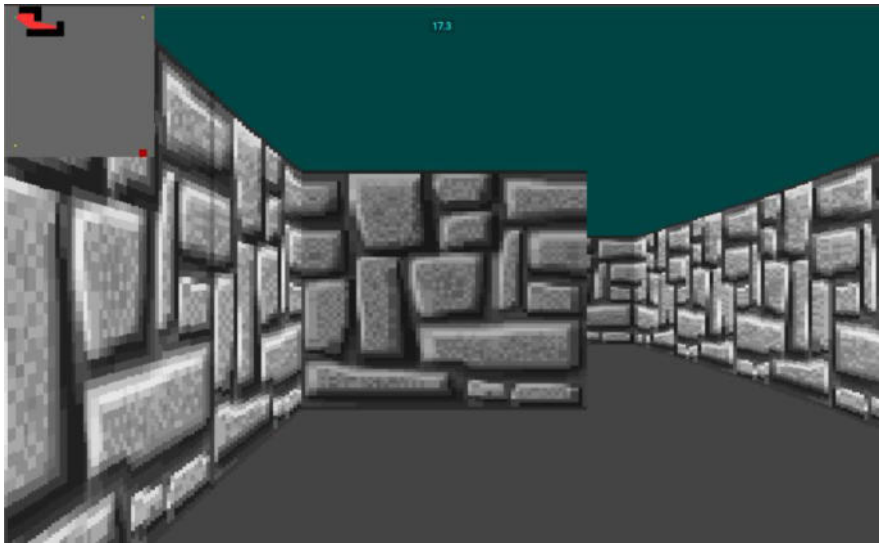

Рис.4.5. Результат 3D проекції з накладанням текстур

Клас гравця (Player) відповідає за промальовування та переміщення гравця на карті, оновлює позиції та кути кидання променів використовує промені для створення проекції відносно положення гравця. Основні властивості:

- ctx – це контекст ігрового полотна, де буде розміщений гравець;
- currentLevel – це посилання ігровий рівень;
- x, y – координати гравця на ігровому полотні;
- moveState – стан руху гравця, приймає значення -1, 0, 1 рух назад, нерухомий та рух вперед відповідно
- rotateState – стан повороту гравця, приймає значення -1, 0, 1 поворот проти годинникової стрілки, статичний стан, поворот за годинниковою стрілкою;
- rotateAngle – кут повороту гравця;
- moveVelocity, rotateVelocity – швидкості руху та повороту відповідно;
- rays – масив променів, що створюються при початковій ініціалізації гравця;
- rayCount – кількість променів;
- eyesLevel – рівень очей гравця;
- doesEyesLevelChangingUp – чи рівень очей рухається вгору.

Та основні методи:

- draw – відповідає за малювання гравця на карті;

- castRays – використовує промені, промальовує їх на карті;
- renderRays – проектує результат кидання променів на ігрове полотно;
- updatePosition – займається оновленням позиції та куту повороту гравця.
- tick – об’єднює методи draw, castRays, renderRays та updatePosition;
- isCollision – використовує посилання на рівень та викликає його метод для перевірки колізії (зіткнення зі стіною);
- forward, moveStateZero, back – методи, що змінюють стан руху;
- left, rotateStateZero, right – методи, що змінюють стан повороту;
- getPlayerData – повертає детальну інформацію гравця, для його збереження відновлення в певний момент гри;
- isOnFinish – сигналізує про досягнення гравцем кінця лабіринту.

Скомбінувавши ці три класи, отримано основу ігрового рушія для створення псевдо-3D графіки. В додатку Б розміщено код цих класів.

Нижче наведено код кидання променя з поступовим розбором вмісту:

В даному фрагменті перезаписуються всі необхідні нам змінні для обчислень

```
...
 this.xIntercept = 0;
 this.yIntercept = 0;
 this.xStep = 0;
 this.yStep = 0;
 this.isLookingDown = this.angle < Math.PI;
 this.isLookingLeft = this.angle > Math.PI / 2 && this.angle < (3 * Math.PI) / 2;
 const angleTan = Math.tan(this.angle);
...
```

Далі обчислюється горизонтальне зіткнення, спочатку береться сусідня клітинка, з тієї сторони, в яку пускається промінь, далі поступово в циклі перевіряється умова зіткнення

```
...
const yTemp = this.y / this.level.tileSize;
 this.yIntercept =
   (this.isLookingDown ? Math.ceil(yTemp) : Math.floor(yTemp)) * this.level.tileSize;
```

```
 this.xIntercept = this.x + (this.yIntercept - this.y) / tan; //neighbor tile
 this.yStep = this.isLookingDown ? this.level.tileSize : -this.level.tileSize;
 this.xStep = this.yStep / tan; //calculating X step according to angle
 let nextXHorizontal = this.xIntercept;
 let nextYHorizontal = this.yIntercept;
 !this.isLookingDown && --nextYHorizontal;
 let isCollidingHorizontal = false;
 while (
   !isCollidingHorizontal &&
   nextXHorizontal >= 0 &&
   nextXHorizontal <= canvasWidth &&
   nextYHorizontal >= 0 &&
   nextYHorizontal <= canvasHeight
 ) {
   const xCollision = (nextXHorizontal / this.level.tileSize) ^ 0; //j for level array
   const yCollision = (nextYHorizontal / this.level.tileSize) ^ 0; //i for level array

   if (
     this.level.isCollision(xCollision, yCollision) ||
     yCollision > this.level.tilesCount - 1
   ) {
     isCollidingHorizontal = true;
     this.wallHitXHorizontal = nextXHorizontal;
     this.wallHitYHorizontal = nextYHorizontal;
   } else {
     nextXHorizontal += this.xStep;
     nextYHorizontal += this.yStep;
   }
 }
...
```

Далі за тим самим принципом обчислюється вертикальне зіткнення:

```
...
let tempX = this.x / this.level.tileSize;
 this.xIntercept =
   (this.isLookingLeft ? Math.floor(tempX) : Math.ceil(tempX)) * this.level.tileSize;
//tile x coord
 this.yIntercept = this.y + (this.xIntercept - this.x) * tan;
 this.xStep = this.isLookingLeft ? -this.level.tileSize : this.level.tileSize;
 this.yStep = this.xStep * tan;
 let nextXVertical = this.xIntercept;
 let nextYVertical = this.yIntercept;
 this.isLookingLeft && --nextXVertical;
 let isCollidingVertical = false;

 while (
   !isCollidingVertical &&
   nextXVertical >= 0 &&
   nextXVertical <= canvasWidth &&
   nextYVertical >= 0 &&
```

```
    nextYVertical <= canvasHeight
  ) {
    const xCollision = (nextXVertical / this.level.tileSize) ^ 0;
    const yCollision = (nextYVertical / this.level.tileSize) ^ 0;

    if (
      this.level.isCollision(xCollision, yCollision) ||
      xCollision > this.level.tilesCount - 1
    ) {
      isCollidingVertical = true;
      this.wallHitXVertical = nextXVertical;
      this.wallHitYVertical = nextYVertical;
    } else {
      nextXVertical += this.xStep;
      nextYVertical += this.yStep;
    }
  }
...
```

Далі шукаються відстані до наших зіткнень:

```
...
// find distance
 let horizontalDist = Number.MAX_VALUE;
 let verticalDist = Number.MAX_VALUE;

 if (isCollidingHorizontal) {
   horizontalDist = calculateDistance(
     this.x,
     this.y,
     this.wallHitXHorizontal,
     this.wallHitYHorizontal
   );
 }
 if (isCollidingVertical) {
   verticalDist = calculateDistance(
     this.x,
     this.y,
     this.wallHitXVertical,
     this.wallHitYVertical
   );
 }
…
```

Далі відстані співставляються, обирається коротша, записується результат та обирається текстура для майбутньої проекції:

```
...
 this.isHittingHorizontal = horizontalDist < verticalDist;
 if (this.isHittingHorizontal) {
```

```
    this.wallHitX = this.wallHitXHorizontal;
    this.wallHitY = this.wallHitYHorizontal;
    this.distanceToWall = horizontalDist;
    this.texturePixel =
      ((this.wallHitX - ((this.wallHitX / this.level.tileSize) ^ 0) *
  this.level.tileSize) /
        this.level.tileSize) *
      textureDim;
  } else {
    this.wallHitX = this.wallHitXVertical;
    this.wallHitY = this.wallHitYVertical;
    this.distanceToWall = verticalDist;
    this.texturePixel =
      ((this.wallHitY - ((this.wallHitY / this.level.tileSize) ^ 0) *
  this.level.tileSize) /
        this.level.tileSize) *
      textureDim;
  }

  this.level.clearMaskTile(this.wallHitX, this.wallHitY, this.isLookingDown,
  this.isLookingLeft);
  this.texturePixel = this.texturePixel ^ 0;
  this.textureId = this.level.getTile(this.wallHitX, this.wallHitY) - 1;
 }
```

4.2.2. Бізнес логіка програми

Для збереження інформації було розроблено наступні ендпойнти:

- /api/login (POST) – обробка авторизації, приймає логін та пароль;
- /api/register (POST) – обробка запиту реєстрації, приймає логін, пароль користувача та зберігає їх в колекції;
- /api/save-game/:userId (POST) – використовується для збереження стану гри. Приймає детальні дані про рівень, гравця та час проходження рівня;
- /api/load-game/:id/:userId (GET) – з його допомогою отримується конкретний запис збереженого стану гри за його ключем та ідентифікатором користувача;
- /api/game-saves/:userId (GET) – повертає список наявних збережень користувача;
- /api/user-exists/:id (GET) – перевіряє існування користувача в системі.

Нижче представлений код обробників запитів на збереження та відновлення стану гри:

```
app.post('/api/save-game/:userId', async (req, res) => {
  res.status(200);
  const { userId } = req.params; if (req.body && userId) {
    const collection = await db.collection('game-saves');
    let response = await collection.insertOne({ ...req.body, userId: userId }); if
    (response) { res.send({ gameSaveId: response.insertedId }); return; }
  }
  res.send(null);
});
app.get('/api/load-game/:id/:userId', async (req, res) => {
  res.status(200);
  const { id, userId } = req.params; if (id && userId) {
    const collection = await db.collection('game-saves');
    let response = await collection.findOne({ _id: ObjectId(id), userId: userId }); if
    (response) { res.send(response); return; }
  }
  res.send(null);
});
```

4.3. Демонстрація результатів

Одне з перших найважливіших вікон гри, є вікно створення рівня, де користувач має змогу обрати розмірність та тип текстурування рівня.

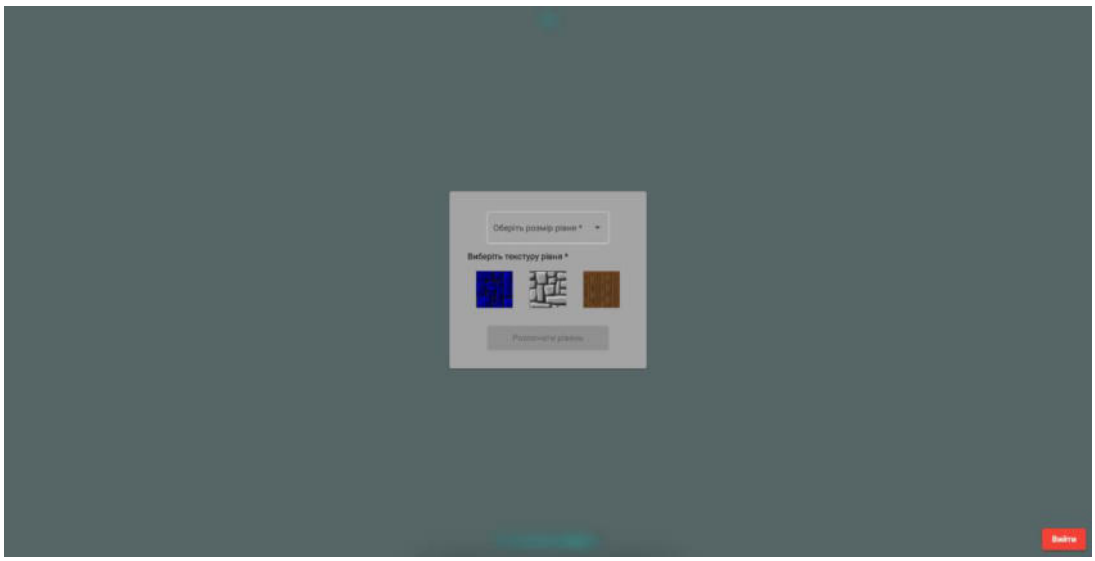

Рис.4.6. Вікно вибору параметрів для запуску рівня

Після генерації користувач попадає на сам рівень і може розпочати гру. На ігромвому полотні відображається наступна інформація:

- Мінікарта в лівому верхньому куті, яка відображає поточну позицію гравця, позицію ключів та клітинку виходу (червоним кольором, так як ключі ще не було зібрано);
- Вгорі посередині відображається час проходження даного рівня;

- Знизу відображається ціль користувача - збір ключів;
- Справа знизу відображається кількість кадрів за секунду.

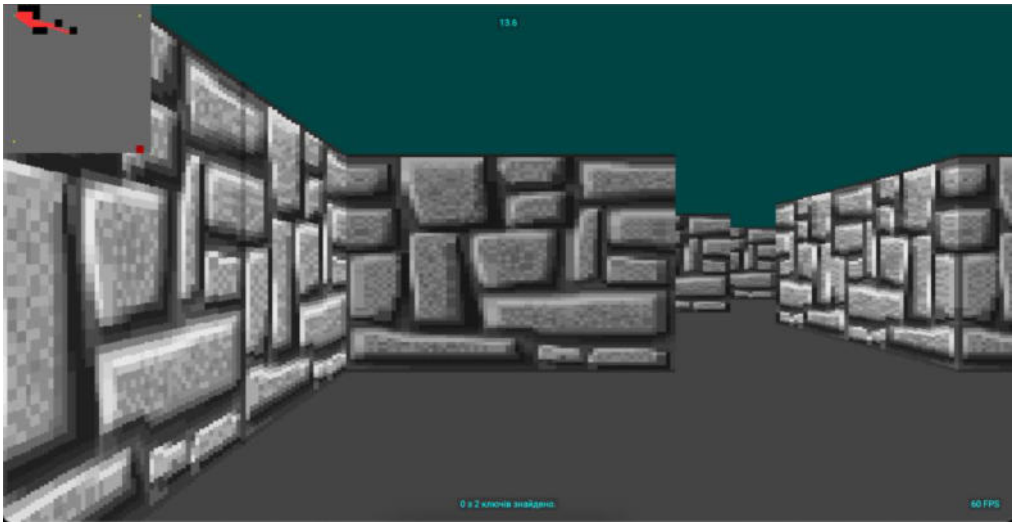

Рис.4.7. Вікно з ігровим полотном

Аби вихід з рівня відкрився, користувач має виконати своє завдання - зібрати ключі.

Рис.4.8. Відображення ключа на ігровому рівні

Лише після виконання завдання в повній мірі вихід відкриється, це буде відзначено як і на мінікарті, так і на самому рівні

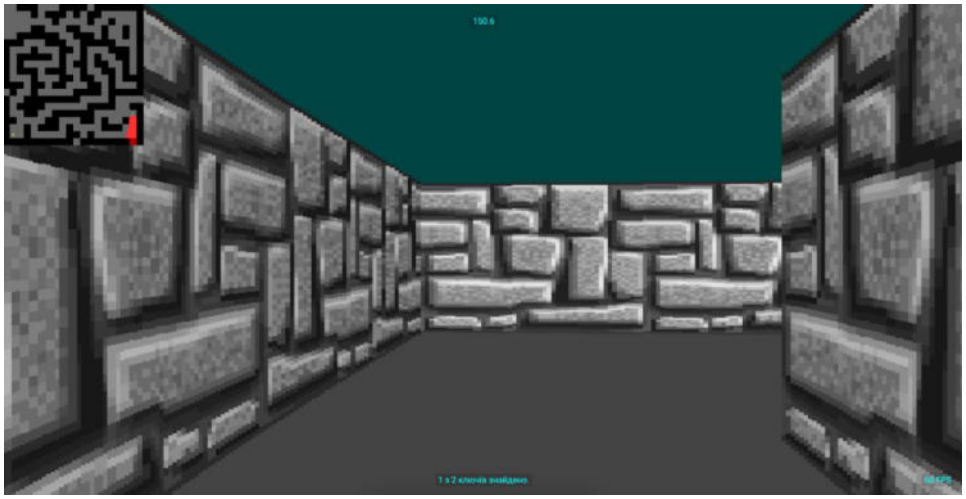

Рис.4.9. Заблокований вихід, який ніяк не відрізняється від звичайної стіни

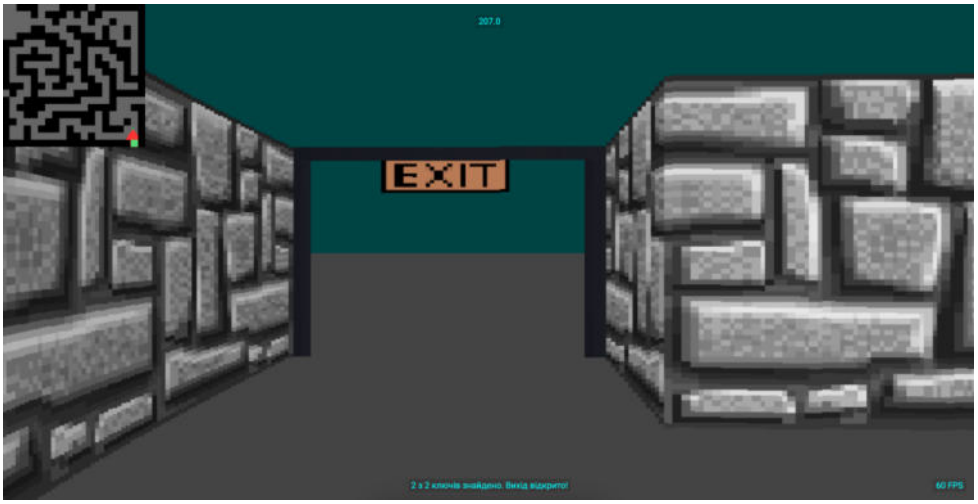

Рис.4.10. Відкритий вихід з лабіринту

В будь-який момент часу користувачеві доступне ігрове меню, за допомогою якого можна:

- Повернутись до гри;
- Розпочати новий рівень;
- Завантажити та зберегти гру;
- Вийти.

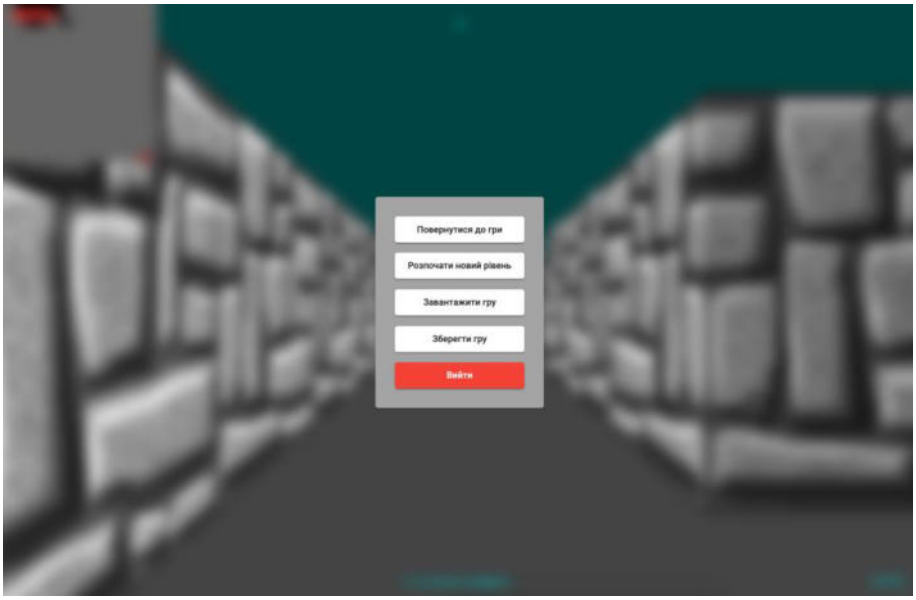

Рис.4.11. Ігрове меню

Меню завантаження ігор відображає детальну інформацію про час збереження, розмірність лабіринту та час проходження рівня

Рис.4.12. Меню завантаження гри

ВИСНОВКИ

Під час виконання даної магістерської роботи я розглянув та описав основні жанри комп'ютерних відеоігор, встановив мету розробки системи. Далі описав загальні характеристики та склав детальну специфікацію вимог до програмного забезпечення.

Далі я описав майбутню архітектуру гри, методи для підтримки чистоти коду, для його подальшої легкої модифікації та зручності тестування. Описав основні інструменти розробки.

Далі я спроектував поведінку системи, створив блок схеми основних алгоритмів та розробив прототипи користувацького інтерфейсу, які надалі використовувалися безпосередньо під час розробки програмного продукту.

Реалізувавши продукт я провів детальний опис головних класів системи, їх властивостей та методів, вказавши дані, з якими вони працюють та результати їхнього виконання. Створене програмне забезпечення надає наступні можливості користувачеві:

- Можливість реєстрації, авторизації чи гри як гість;
- Випадкова генерація рівнів;
- Проекція 2D простору в 3D за допомогою рейкастингу
- Вільна навігація користувача по ігровому рівні;
- Можливість збереження та відновлення певного моменту гри;
- Збір предметів-ключів для забезпечення відкритого виходу з рівня.

СПИСОК ЛІТЕРАТУРИ

1. Ray casting – [Електронний ресурс]. – Режим доступу: https://permadi.com/1996/05/ray-casting-tutorial-table-of-contents

2. Бойовик (жанр відеоігор) – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Бойовик_(жанр_відеоігор)

3. Стратегічна відеогра – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Стратегічна_відеогра

4. Рольова відеогра – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Рольова_відеогра

5. Симуляційна відеогра – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Симуляційна_відеогра

6. Пригодницька відеогра – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Пригодницька_відеогра

7. SOLID principle – [Electronic resource]. – Web page: https://en.wikipedia.org/wiki/SOLID

8. DRY principle – [Electronic resource]. – Web page: https://en.wikipedia.org/wiki/Don’t_repeat_yourself

9. Angular framework documentation – [Electronic resource]. – Web page: https://angular.io/docs

10. Singleton pattern – [Electronic resource]. – Web page: https://refactoring.guru/design-patterns/singleton

11. RxJS overview – [Electronic resource]. – Web page: https://rxjs.dev/guide/overview

12. NodeJS documentation – [Electronic resource]. – Web page: https://nodejs.org/en/docs/

13. MongoDB documentation – [Electronic resource]. – Web page: https://docs.mongodb.com/manual/

14. Створення комп’ютерної гри з використанням технології рейкастингу - [Електронний ресурс]. – Режим доступу:

http://www.konferenciaonline.org.ua/ua/article/id-515/

15. Ray Casting for Modeling Solids, Computer Graphics and Image / Scott D, Roth – 1982. – p. 109-144.

16. Black Art of 3D Game Programming / LaMothe, Andre – 1995. – p. 14, 398, 935-936, 941-943.

17. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing / Woop, Sven; Schmittler, Jörg; Slusallek, Philipp – 2005. – p. 2