БАКАЛАВРСЬКА РОБОТА

БР. ІП - 47.00.00.000 ПЗ

Група ІП-21-3

Ковальова Анастасія

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ковальова Анастасія Сергіївна

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Реалізація клієнтської стратегії ігрового додатку

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Ковальова А.С.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Ваврик Тетяна Олександрівна, асистент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Ковальовій Анастасії Сергіївні

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) **“ Реалізація клієнтської стратегії ігрового додатку ”**

керівник проекту (роботи) Ваврик Тетяна Олександрівна, асистент

затверджені наказом закладу вищої освіти від **“** 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 09 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз побудови кросплатформних ігрових додатків
2. Реалізація системної архітектури ігрового додатку
3. Діаграма пакетів та взаємозв'язки компонентів
4. Процедури рендерингу 3D-сцени у WebGL
5. Програмна реалізація клієнтської стратегії ігрового додатку

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

1. Діаграма робочого процесу розгортання (рис. 2.1)
2. Діаграма залежностей (рис. 2.2)
3. Діаграма пакетів (рис. 2.3)
4. Схема робочого процесу (рис. 2.4)
5. Перший екран історії після переходу JQuery (рис. 3.8)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз побудови кросплатформних ігрових додатків	04.05.2025	виконано
2	Реалізація системної архітектури ігрового додатку	15.05.2025	виконано
3	Діаграма пакетів та взаємозв'язки компонентів	21.05.2025	виконано
4	Процедури рендерингу 3D-сцени у WebGL	28.05.2025	виконано
5	Програмна реалізація клієнтської стратегії ігрового додатку	03.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	09.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 75 сторінок, 25 рисунків, список використаних джерел із 31 найменуваннями.

Метою роботи є проєктування та реалізація клієнтської стратегії кросплатформного ігрового додатку головоломки на основі сучасних веб-технологій з урахуванням збереження стану гри, ефективного рендерингу графіки та забезпечення інтерактивного користувацького досвіду.

Об'єкт дослідження - процес розробки клієнтської частини кросплатформного ігрового додатку.

Предмет дослідження - методи, засоби та архітектурні підходи до реалізації клієнтської стратегії у веб-орієнтованих ігрових додатках.

В першому розділі визначено ключові принципи та виклики, пов'язані з проєктуванням ігрових додатків, розкрито сюжетну основу гри та вимоги до її реалізації.

В другому розділі розроблено архітектуру клієнтської частини, інтегровано бібліотеки та побудовано структурну модель програмних компонентів.

В третьому розділі реалізовано повнофункціональний клієнтський модуль гри, забезпечено управління станом, графікою, інтерфейсом та протестовано готовий продукт

Висновок: реалізовано клієнтську стратегію управління ігровим процесом у веб-орієнтованому додатку головоломки, що поєднує механізми збереження стану гри, гнучку ігрову логіку, графічну інтеграцію через WebGL та адаптивний користувацький інтерфейс

КЛЮЧОВІ СЛОВА: КЛІЄНТСЬКА СТРАТЕГІЯ, ВЕБ-ГРА, ГОЛОВОЛОМКА, КРОСПЛАТФОРМНІСТЬ, WEBGL, JAVASCRIPT, ЛОКАЛЬНЕ СХОВИЩЕ, ІГРОВИЙ ІНТЕРФЕЙС, АРХІТЕКТУРА ДОДАТКУ, РЕНДЕРИНГ, ІНВЕНТАР, UX/UI.

ANNOTATION

The bachelor's thesis contains 75 pages, 25 figures, a list of used sources with 31 names.

The purpose of the work is to design and implement the client strategy of a cross-platform puzzle game application based on modern web technologies, taking into account the preservation of the game state, effective graphics rendering and ensuring an interactive user experience.

The object of the study is the process of developing the client part of a cross-platform game application.

The subject of the study is methods, tools and architectural approaches to implementing the client strategy in web-based game applications.

The first section identifies key principles and challenges related to the design of game applications, reveals the plot basis of the game and the requirements for its implementation.

In the second section, the architecture of the client part is developed, libraries are integrated and a structural model of software components is built.

In the third section, a fully functional client game module is implemented, state, graphics, and interface management are provided, and the finished product is tested.

Conclusion: a client strategy for controlling the gameplay in a web-based puzzle application is implemented, combining game state saving mechanisms, flexible game logic, graphical integration via WebGL, and an adaptive user interface.

KEYWORDS: CLIENT STRATEGY, WEB GAME, PUZZLE, CROSS-PLATFORM, WEBGL, JAVASCRIPT, LOCAL STORAGE, GAME INTERFACE, APPLICATION ARCHITECTURE, RENDERING, INVENTORY, UX/UI.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

2D - 2-вимірна координатна графічна система, на якій розташування визначається віссю x та y.

Cookies - метадані, збережені на клієнті, які дозволяють програмістам відстежувати інтернет-трафік користувача.

DOM - Об'єктна модель документа - структура батьківсько-дочірнього представлення веб-сторінки документа.

GIMP - Програма GNU для редагування зображень, яка надає інструменти для модифікації існуючих зображень.

Графічний конвеєр - Термін, що описує етапи, через які проходить 3D-графіка, щоб стати 2D-растрованими зображеннями.

IOS - Операційна система для смартфонів та планшетів, розроблена компанією Apple.

Javascript - Веб-мова, призначена для динамічної зміни статичних html-елементів.

Браузер - програма, яка забезпечує зв'язок через Інтернет, зазвичай через HTTP-з'єднання між клієнтом та сервером.

JQUERY - Об'єктно-орієнтована бібліотека, побудована на основі JavaScript.

JQUERYUI - Побудована на основі JQUERY. Вона призначена для надання функціональності переміщення та скидання елементам.

NPC - Non-Playable Character - Неігровий персонаж - зазвичай персонаж у відеогрі, який надає гравцям інформацію про гру або квести.

OPENGL - Мова бібліотеки графіки з відкритим вихідним кодом, яка дозволяє рендеринг графіки на графічній карті.

ШЕЙДЕР - програма, яка маніпулює окремими графічними елементами, такими як вершина або піксель.

TOUCH PUNCH - Бібліотека, яка перетворює події миші на події дотику, сумісні з портативними пристроями.

WebGL - Реалізація популярної бібліотеки openGL за стандартами веб.

ВСТУП

Індустрія цифрових ігор є однією з найдинамічніших галузей сучасних інформаційних технологій, що стрімко розвивається в умовах широкого впровадження мобільних пристроїв, зростання обчислювальних можливостей браузерів і популяризації кросплатформних рішень. Ігрові додатки сьогодні — це не лише засіб розваги, а й потужний інструмент соціальної взаємодії, навчання, психоемоційного впливу, а також майданчик для реалізації інноваційних підходів у програмуванні, дизайні та штучному інтелекті. Серед численних жанрів, особливу увагу привертають головоломки, які поєднують у собі інтелектуальні виклики, логічне мислення та динамічну взаємодію користувача з ігровим середовищем.

У процесі створення таких додатків надзвичайно важливою є ефективна реалізація клієнтської частини — програмного забезпечення, що відповідає за обробку взаємодій користувача, візуалізацію контенту, збереження ігрового стану, керування рівнями та забезпечення зручного інтерфейсу. Від якості клієнтської стратегії залежить продуктивність, стабільність та загальне враження користувача від гри. Саме тому дослідження і реалізація клієнтських компонентів у веб-іграх, що працюють у браузерному середовищі без необхідності встановлення, є важливим та актуальним напрямом прикладної інформатики.

Зі зростанням популярності кросплатформних технологій, особливо важливим стає розроблення додатків, які однаково ефективно функціонують як на мобільних пристроях, так і на десктопах. У цьому контексті веб-орієнтовані технології, зокрема JavaScript, WebGL, jQuery, HTML5 та локальне сховище, надають широкі можливості для створення складних інтерактивних систем, що підтримують 3D-графіку, збереження прогресу та адаптивний користувацький інтерфейс.

У дипломній роботі розглянуто повний цикл створення клієнтської частини гри-головоломки з елементами сюжету, інвентаризації, рівнів складності та збереження стану гри. У межах дослідження проаналізовано архітектуру, побудовано програмну модель, обґрунтовано вибір технологій, реалізовано основні функціональні модулі, а також проведено тестування для оцінки стабільності та зручності користування.

Запропонована реалізація клієнтської стратегії може бути основою для подальшого розвитку повноцінного комерційного або навчального продукту, а також прикладом для навчання розробників сучасних веб-ігрових додатків. Досвід, здобутий у процесі роботи, демонструє важливість інтеграції знань з програмної інженерії, веб-розробки, UI/UX-дизайну та роботи з графічними активами.

Актуальність роботи

Актуальність теми обумовлюється постійним зростанням попиту на кросплатформні веб-ігри, що не потребують встановлення та дозволяють охоплювати широку аудиторію. Розробка головоломок в онлайн-середовищі потребує ефективної реалізації клієнтської логіки, що включає обробку сценаріїв взаємодії, збереження прогресу, гнучку роботу з графічними елементами та підтримку користувацької персоналізації. У цьому контексті особливої важливості набувають дослідження ефективного управління клієнтською частиною, використання сучасних бібліотек (WebGL, jQuery) та оптимізація UX/UI. Запропонована розробка дозволяє не лише вирішити низку прикладних завдань, а й створити платформу для подальших досліджень у сфері гейміфікації, навчальних середовищ та взаємодії у віртуальних просторах.

Метою роботи є проєктування та реалізація клієнтської стратегії кросплатформного ігрового додатку головоломки на основі сучасних веб-технологій з урахуванням збереження стану гри, ефективного рендерингу графіки та забезпечення інтерактивного користувацького досвіду.

Завдання дослідження

1. Проаналізувати особливості розробки кросплатформних головоломок для мобільних та веб-пристроїв.

2. Визначити системні вимоги до клієнтської частини ігрового додатку.

3. Розробити архітектуру та модель компонентів клієнтської частини гри.

4. Реалізувати механізми збереження прогресу, управління рівнями та інвентарем.

5. Інтегрувати графічні бібліотеки для відтворення візуальних активів та 3D-графіки.

6. Створити інтерфейс користувача для основних екранів гри.

7. Провести тестування реалізованої клієнтської стратегії та визначити напрями подальшого розвитку.

Об’єкт дослідження - процес розробки клієнтської частини кросплатформного ігрового додатку.

Предмет дослідження - методи, засоби та архітектурні підходи до реалізації клієнтської стратегії у веб-орієнтованих ігрових додатках.

Методи дослідження:

- аналітичний метод (для вивчення сучасних технологій клієнтської розробки);
- метод системного аналізу (для проєктування архітектури додатку);
- моделювання (для створення ігрової логіки та взаємодії компонентів);
- експериментальний метод (для реалізації програмного прототипу);
- метод тестування (для перевірки стабільності функціонування гри).

Наукова новизна

Наукова новизна роботи полягає в реалізації узагальненої клієнтської стратегії управління ігровим процесом у веб-орієнтованому додатку головоломки, що поєднує механізми збереження стану гри, гнучку ігрову

логіку, графічну інтеграцію через WebGL та адаптивний користувацький інтерфейс.

Практичне застосування

Результати дипломної роботи можуть бути використані для створення повнофункціональних браузерних ігор, навчальних симуляторів, гейміфікованих платформ, а також як методична база для підготовки фахівців у сфері веб-розробки та ігрового дизайну.

Бакалаврська робота містить 75 сторінок, 25 рисунків, 3 розділи список використаних джерел із 31 найменуванням.

РОЗДІЛ 1. АНАЛІЗ ПОБУДОВИ КРОСПЛАТФОРМНИХ ІГРОВИХ ДОДАТКІВ

1.1. Аналіз та рішення розробки кросплатформних ігор-головоломок для мобільних пристроїв

Зважаючи на обмежені обчислювальні можливості мобільних пристроїв, розробка ігор для цієї платформи вимагає оптимізації графічних ресурсів при збереженні високого рівня ігрової привабливості та зручності використання. Незважаючи на те, що ігри-головоломки традиційно не є основними для мобільних пристроїв, вони демонструють значний потенціал завдяки своїй адаптивності до цих платформ. Останніми роками цей жанр набув значної популярності та визнання, про що свідчить успіх таких ігор, як Myst, The Seventh Guest та Portal, які залучають гравців складними головоломками та винагороджують їх наративними елементами після успішного виконання завдань.

1.1.1. Проблематика та виклики

Розробка подібних ігор для мобільних пристроїв зіткнулася з низкою проблем, головним чином пов'язаних з фрагментацією операційних систем. Різні платформи, такі як Android, iOS та Windows Phone, використовують унікальні набори для розробки програмного забезпечення (SDK), що призводить до несумісності кінцевих продуктів між різними екосистемами. Ця ситуація ускладнює та здорожує процес розробки, вимагаючи створення окремих версій гри для кожної платформи.

Перспективним рішенням для подолання проблеми кросплатформності є використання браузерних технологій. Оскільки всі сучасні цифрові пристрої оснащені веб-браузерами, використання вебу як платформи для розповсюдження ігор дозволяє досягти універсальної сумісності.

1.1.2. Проектна реалізація та використані технології

Метою цього проекту було створення кросплатформної гри-головоломки, здатної функціонувати на будь-якому цифровому пристрої без необхідності встановлення додаткових плагінів. Для досягнення цієї мети були використані наступні веб-технології:

jQuery: Ця бібліотека забезпечує функціональність перетягування та скидання (drag-and-drop), що є ключовим елементом взаємодії в багатьох іграх-головоломках.

Touch Punch: Доповнення до jQuery, яке дозволяє адаптувати функції, призначені для взаємодії з мишею, до сенсорного інтерфейсу, що є критично важливим для мобільних пристроїв.

Локальне сховище (Local Storage): Надає можливість зберігання даних на пристрої користувача, усуваючи необхідність у серверній взаємодії для збереження прогресу гри.

WebGL: ця веб-технологія забезпечує рендеринг інтерактивної 3D-графіки безпосередньо в браузері, дозволяючи виконувати обробку графіки на планшеті або телефоні користувача через веб-команди, оптимізуючи таким чином використання локальних ресурсів.

Завдяки інтеграції цих технологій було створено гнучке та доступне ігрове рішення, що відповідає сучасним вимогам кросплатформної розробки для мобільних пристроїв.

1.2. Цілі, методологія та характеристики проекту розробки веб-орієнтованої гри-головоломки

Метою цього проєкту є розробка клієнтської веб-орієнтованої точкової гри-головоломки з інтегрованою сюжетною лінією. Використання веб-стандартів зумовлене прагненням створити відносно легковагову гру, яка не вимагає спеціалізованого апаратного забезпечення чи програмних платформ.

Крім того, орієнтація на веб-середовище забезпечує широке поширення та сумісність з різноманітними операційними системами.

1.2.1. Обсяг проєкту

Цей проєкт обмежується виключно клієнтською розробкою, виключаючи використання серверного програмування або створення нативних застосунків для мобільних операційних систем, таких як Android, iOS або Windows Phone.

Гра належить до жанрів головоломки та пригод.

1.2.2. Системні вимоги

Розроблена гра орієнтована на універсальну сумісність з усіма типами пристроїв. Вона призначена для функціонування на всіх поширених операційних системах персональних комп'ютерів (Windows, Linux, macOS тощо), а також на операційних системах мобільних телефонів і планшетів (Android, WebOS, Windows Phone тощо).

1.2.3. Ігрова механіка

Основний ігровий процес зосереджений на дослідженні та навігації двовимірним світом, що містить елементи стародавньої технології. Взаємодія гравців з ігровими елементами реалізується за допомогою функції перетягування (drag-and-drop). Гравці можуть обирати об'єкти та маніпулювати ними за допомогою кліку, а потім переміщувати їх у необхідні локації.

Поточна версія гри має орієнтовну тривалість проходження від п'ятнадцяти до тридцяти хвилин, проте у повній запланованій версії передбачається значне збільшення часу проходження, що становитиме щонайменше чотири-п'ять годин.

1.3. Опис ігрової тематики

1.3.1. Ігрова історія: Передумови

У футуристичному сценарії, що становить основу наративу, людство сприймає Інтернет як всеосяжне сховище знань. Кожна людина має імплантований у мозок чіп, який дозволяє завантажувати спогади, думки та досвід у режимі реального часу. Початкова мета цієї технології полягала в демократизації знань та стимулюванні генерації справді нових ідей.

Проте, така повсюдність знань та гіперзв'язок виявилися надмірним тягарем для людської психіки. Приватні резиденції трансформувалися у своєрідні "технологічні опіумні притони", оскільки люди стали залежними від життя та досвіду інших. Ця залежність призвела до деградації особистісного життя та активності, оскільки індивіди віддавали перевагу опосередкованому досвіду над власним. Навіть ті, хто уникнув психологічної залежності, страждали від фізичних наслідків багаторічного зловживання технологіями, що проявлялося в ожирінні та серцево-судинних захворюваннях. Особи, здатні робити внесок у суспільство, виявлялися перевантаженими зростаючим обсягом інформації, що призводило до масових випадків психічних розладів. Весь цей час кожна думка та досвід ретельно документувалися.

Зрештою, людство змушене було зіткнутися з наслідками тотальної діджиталізації. Нечисленні вцілілі, яким вдалося відключитися від Мережі, виявилися неспроможними функціонувати у світі без технологій, що призвело до повернення у "кам'яний вік".

Через тисячу років після описаних подій ми бачимо, що людство почало освоювати старі технології, проте все ще перебуває на стадії "темних віків" розвитку. Знання про залежність їхніх предків від комп'ютерів мінімальні, а єдині збережені спогади трансформувалися у забобонні застереження. Релігійні лідери поширюють догмати про те, що в стародавніх

технологіях перебуває злий дух, і що технологічних імперій минулого слід уникати.

1.3.2. Опис головного героя

У дитинстві головний герой, Аттік, мешкає поблизу технологічних руїн великого міста. Ці руїни визначені як заборонена зона, проте з плином часу рівень безпеки знижується.

Внаслідок виклику Аттік опиняється у місті, яке нагадує лабіринт, оскільки воно було спроєктоване для взаємозв'язку, а не для людського пересування. Прокладаючи шлях через темні та покинуті алеї, він зазнає падіння та ламає ногу. У стані безвихідності та неспроможності знайти вихід, старі технології перезавантажуються і демонструють йому голографічний спогад. Цей спогад деталізує нанотехнологічну зброю, яка здійснює лікування його зламаної ноги. Травмований, Аттік витісняє цей інцидент зі свідомості.

Через роки чума охоплює його село. Аттік безпорадно спостерігає за загибеллю сімей, але лише коли хвороба досягає його порогу, його витіснені спогади повертаються. З хворобою його дружини та найкращого друга всі його страхи та забобони зникають, і він розглядає подорож у заборонені руїни. Він припускає, що деякі з технологій можуть бути корисними для боротьби з чумою.

1.3.3. Ігрова сюжетна лінія. Занурення в лабіринт

Потрапивши в лабіринт, Аттік виявляє, що застряг. Вхід закривається за ним, і лише вибрані двері відкриваються одна за одною, ведучи його далі в підземелля, здається, за заздалегідь визначеним шляхом.

Надія Аттіка спалахує, коли він зустрічає "і" – свідомість, яка обіцяє допомогти у пошуках ліків. Проте незабаром протагоніст виявляє, що "і" не настільки надійний, як сподівався, оскільки свідомість починає

розколюватися на окремі особистості. Спочатку "і" трансформується в Аврору, молоду дівчину, чиє фізичне життя давно завершилося, але чиї спогади продовжують існувати, а потім миттєво "і" перетворюється на хлопця Аврори, Райлі. Аттік починає усвідомлювати мінливість "і", але продовжує рухатися вперед.

1.4. Керування грою та інтерфейс користувача

Взаємодія користувача з програмним продуктом реалізується через два ключові інтерфейси: екран вибору та сцену. Екран вибору є початковим інтерфейсом, призначеним виключно для активації нової гри або продовження поточної. Введення користувацьких даних на цьому екрані обмежене, а навігація здійснюється за допомогою кнопок для переходу між сторінками або надсилання інформації. Переважна частина взаємодії користувача відбувається зі сценою, яка складається з текстурованого фону, що відображає конкретну кімнату, та інтерактивних об'єктів на передньому плані.

Спосіб взаємодії користувача з цими інтерфейсами залежить від типу використовуваного пристрою. На пристроях з мишею (наприклад, персональних комп'ютерах) вибір елементів на екрані здійснюється лівою кнопкою миші. Переміщення в ігровому просторі та операції перетягування виконуються шляхом переміщення курсору миші. Введення текстових даних здійснюється за допомогою клавіатури. На сенсорних інтерфейсах (наприклад, мобільних телефонах та планшетах) принцип взаємодії відрізняється: дотик до екрану дозволяє вибирати елементи, а рух пальцем по екрану забезпечує функціонал перетягування. У випадках, коли до мобільного пристрою підключена фізична клавіатура, введення тексту здійснюється через неї. В іншому випадку, при дотику до поля введення тексту активується нативна екранна клавіатура пристрою.

Основним чинником, що забезпечує повторюваність ігрового процесу, є сюжетна лінія. Поточна версія гри не передбачає розгалужень сюжету, проте у запланованій кінцевій версії передбачається впровадження множинних шляхів, варіантів альянсів та альтернативних закінчень, що значно підвищить реграбельність.

1.4.1. Сиквел

Розробка сиквелу для поточної гри не планується. Оскільки наратив гри тісно пов'язаний з розгортанням свідомості "i", подальші можливості для дослідження в рамках цього сюжету є обмеженими. Однак, у разі підтвердження високої залученості ігрового процесу, його механіки можуть бути адаптовані для використання в іграх, що розгортаються в інших віртуальних світах.

1.4.2. Візуальні активи

Гра потребує 2D-активів, які будуть використовуватися як фони, кнопки, текстові поля, персонажі та елементи, що перетягуються. Фони мають фіксований розмір, зазвичай 800times400 пікселів, що оптимізовано для заповнення вікна браузера типового користувача. Як показано на Рисунку 1, кнопки та текстові поля мають прямокутну форму з градієнтом від чорного до електрично-синього. Вибір кольорів обумовлений художньою концепцією гри: чорний символізує темний лабіринт, а синій — електричне життя, що населяє ігровий світ.

Рисунок 1.1 - Текстова кнопка, що відображає колірну тему

Персонажі намальовані у стилі аніме, що є характерною ознакою наративних ігор, таких як рольові ігри. На відміну від елементів, що

перетягуються, які створюються шляхом модифікації зображень за допомогою програмного забезпечення для редагування графіки, персонажі повинні виглядати живими та виразними. Прикладом є Аттік, зображений на рисунку 1.2.

Рисунок 1.2 - Зображення Аттіка, що ілюструє аніме-стиль гри

Повне відображення всіх ігрових елементів у дії представлено в третьому розділі. Наприклад, рисунок 1.3 демонструє ігровий рівень з перетягуваним металевим павуком. Як можна помітити, більшість кольорів є темними, з темним павуком та темним фоном.

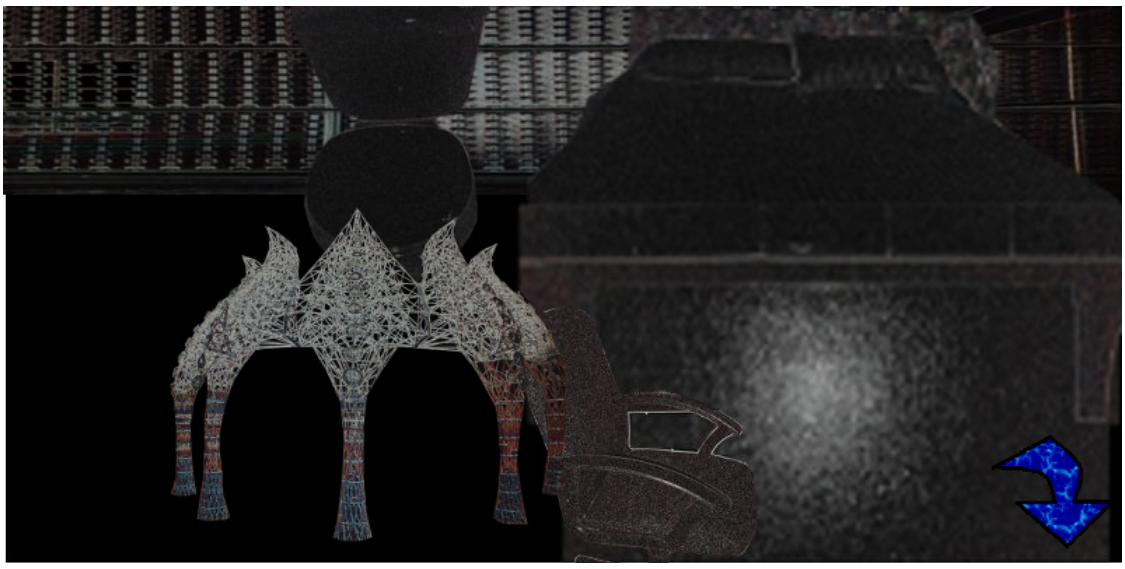

Рисунок 1.3 - Темний фон і світліший елемент, що перетягується

Рисунок 1.4 ілюструє наративний сегмент після завершення виклику. Тут відображені персонажі в аніме-стилі, текстові поля та кнопки з градієнтом від чорного до електрично-синього, а також елементи користувацького інтерфейсу. Для переходу до наступного текстового блоку користувач має натиснути кнопки для переходу вперед або назад.

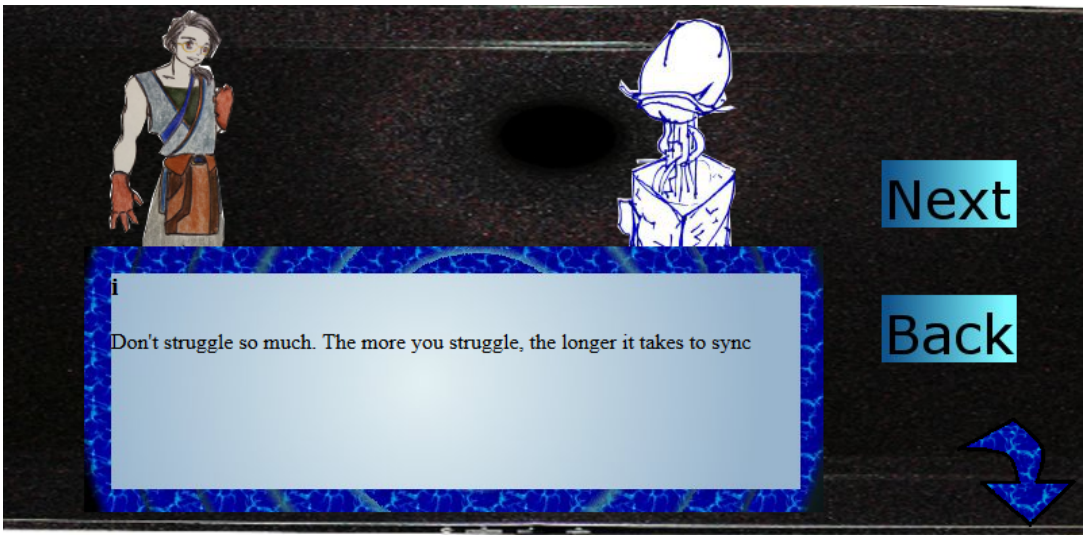

Рисунок 1.4 - Текстові поля та кнопки електрично-синього кольору

Рисунки 1.3 і 1.4 спільно демонструють ще одну особливість користувацького інтерфейсу — можливість перетягувати та скидати елементи. Перший рисунок показує екран до того, як перетягуваний елемент був переміщений до місця призначення, а другий рисунок ілюструє ефект після успішного перетягування. Користувацький інтерфейс розроблений для забезпечення простої взаємодії за принципом "перетягни та скинь".

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СИСТЕМНОЇ АРХІТЕКТУРИ ІГРОВОГО ДОДАТКУ

2.1. Робочий процес розгортання

Взаємодія клієнта з сервером обмежена виключно отриманням веб-сторінок. Всі подальші операції відбуваються на стороні клієнта, незалежно від типу пристрою (персональний комп'ютер, ноутбук, смартфон чи планшет). Цей процес ілюструє рисунок 2.1.

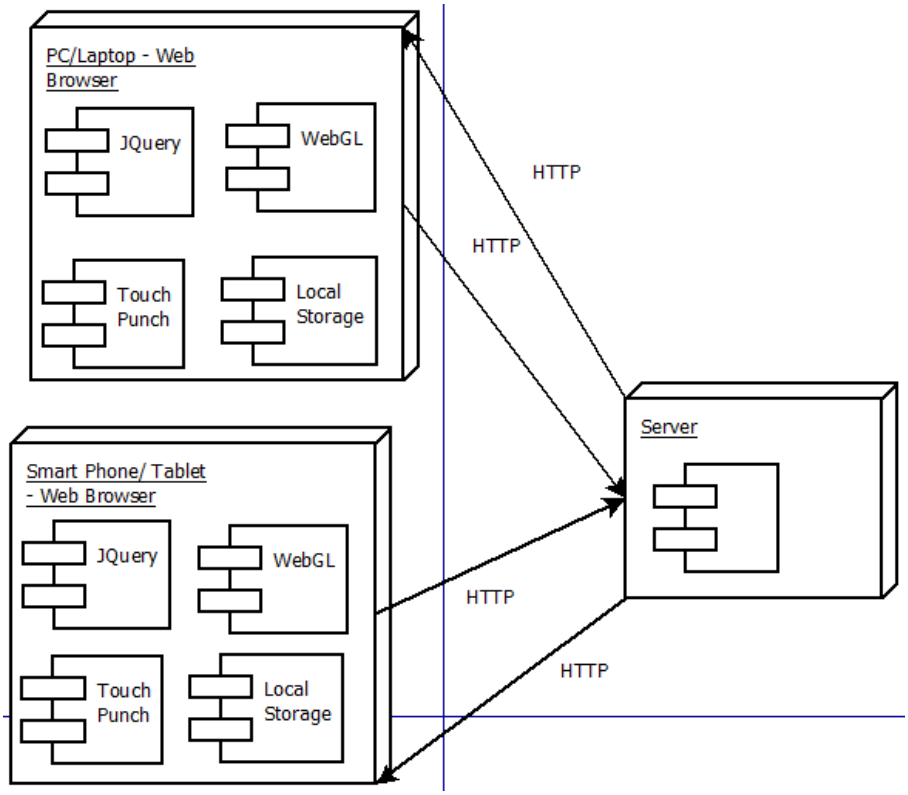

Рисунок 2.1 - Діаграма робочого процесу розгортання

При ініціюванні гри клієнт надсилає HTTP-запит до сервера для отримання необхідного контенту. У відповідь сервер, використовуючи HTTP-з'єднання, передає клієнту вміст сторінки разом з усіма необхідними бібліотеками для виконання гри у браузері клієнта. Вся подальша взаємодія

для даної сторінки відбувається локально на клієнтському пристрої, що усуває потребу в додаткових HTTP-запитах від клієнта до сервера. На відміну від типових веб-додатків, де кожна інтеракція (натискання кнопки, відправлення даних облікового запису тощо) генерує окремий HTTP-запит, у даному проєкті на сторінку припадає лише одна взаємодія з сервером, незалежно від того, використовується мобільний пристрій чи традиційний комп'ютер.

Ця незалежність від сервера досягається шляхом прямого підключення скриптів у розділі заголовка HTML-сторінок. Ці посилання вказують на зовнішні бібліотеки, що зберігаються в окремій директорії. Деякі з цих бібліотек є автономними, тоді як інші посилаються на додаткові бібліотеки в тій же директорії.

2.2. Розробка діаграми залежностей

Рисунок 2.2 використовує файл movingEye.html для ілюстрації цих залежностей, оскільки він є одним з файлів, що інтегрує всі зовнішні бібліотеки. Суцільні лінії позначають пряму залежність HTML-файлу від бібліотеки (у даному випадку, від усіх п'яти бібліотек, оскільки вони включені в заголовок файлу). Порядок виклику бібліотек JavaScript у заголовку визначається їхніми взаємними залежностями, що позначені пунктирними лініями.

Наприклад, Touch Punch залежить від jquery-ui-1.8.23.custom.min.js, який, у свою чергу, залежить від jquery-1.7.2.min.js. Інша взаємозалежність існує між бібліотеками WebGL та jQuery. Файл jquery-1.7.2.min.js має бути викликаний першим, потім glMatrix-0.9.5, а потім webgl-utils.js; отже, стрілки залежності спрямовані від webgl-utils.js до glMatrix-0.9.5 і далі до jquery-1.7.2.min.js.

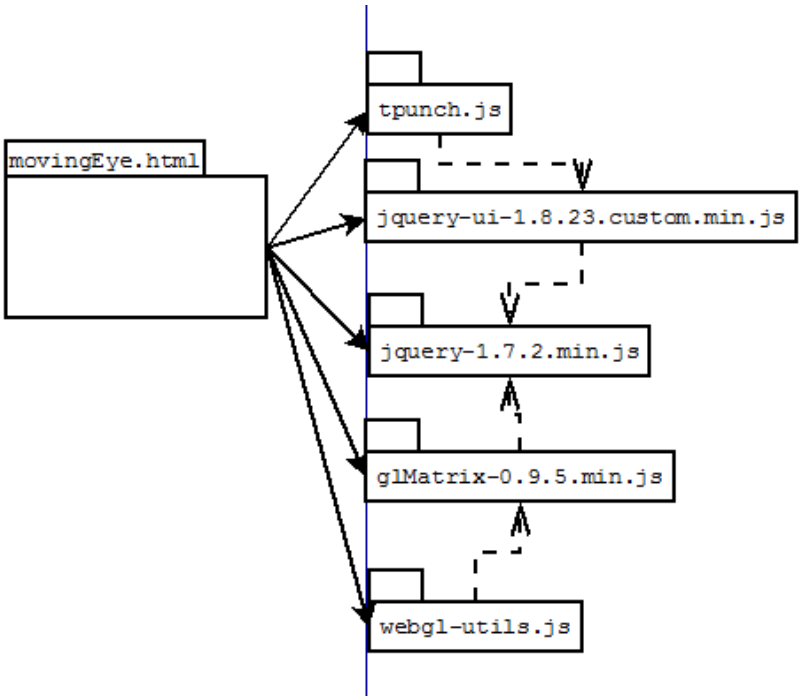

Рисунок 2.2 - Діаграма залежностей

Всі ці зовнішні бібліотеки розроблені для динамічного виконання у браузері без залучення сервера, тому вони повністю реалізовані на JavaScript.

2.3. Інтеграція зовнішніх бібліотек для клієнтської веб-розробки

2.3.1. Бібліотеки JQuery

Після початкового завантаження веб-сторінки welcome.html відбувається розширення нативної функціональності браузера за рахунок інтеграції зовнішніх бібліотек, першою з яких є бібліотека JQuery. JQuery призначений для спрощення використання існуючих можливостей JavaScript. Перша бібліотека, jquery-1.7.2.min.js, є фундаментальною, оскільки вона відповідає за створення об'єктів JQuery, що зумовлює її пріоритетний виклик у розділі <head> HTML-документа:

```
<head>
<script src="script/external_libraries/jquery-1.7.2.min.js"></script>
</head>
```

Друга бібліотека, jquery-ui-1.8.23.custom.min.js, стає необхідною лише після початку гри, коли виникає потреба у функціоналі перетягування (drag-and-drop) елементів. Відповідно, її підключення здійснюється після базової бібліотеки JQuery:

```
<script type="text/javascript" src="script/external_libraries/jquery-1.7.2.mi
<script type="text/javascript" src="script/external_libraries/jquery-ui-1.8.2
```

2.3.2. Touch Punch

Після успішної інтеграції функціональності JQuery відбувається виклик бібліотеки Touch Punch:

```
<script type="text/javascript" src="script/external_libraries/tpunch.js"></sc
```

Ця бібліотека, розташована у тому ж розділі заголовка, де були підключені кореневий файл JQuery та JQuery UI, залежить від функціональності віджетів бібліотеки JQuery. Отже, її виклик відбувається після підключення перших двох бібліотек.

2.3.3. Бібліотеки WebGL

Для реалізації 3D-графіки у проєкті необхідна інтеграція бібліотеки WebGL. Слід зазначити, що одного лише виклику файлу webgl-utils.js недостатньо. На відміну від традиційного OpenGL, WebGL не має вбудованих бібліотек для маніпуляцій з матрицями.

З огляду на значну потребу в матричних операціях у 3D-програмуванні, необхідно підключити зовнішній файл glMatrix-0.9.5.min.js для роботи з матрицями. Функції матриць повинні бути доступні до ініціалізації WebGL, тому послідовність викликів є наступною: спочатку бібліотека матриць, а потім WebGL:

```
<script src="script/external_libraries/jquery-1.7.2.min.js"></script>
<script type="text/javascript" src="script/external_libraries/glMatrix-0.9.5.
<script type="text/javascript" src="script/external_libraries/webgl-utils.js"
<script type="text/javascript" src="script/external_libraries/jquery-ui-1.8.2
<script type="text/javascript" src="script/external_libraries/tpunch.js"></sc
```

2.3.4. Доступ до локального сховища

Функціональність локального сховища (Local Storage) не потребує підключення зовнішніх бібліотек, оскільки вона є стандартною функцією браузерів з підтримкою HTML5. З цієї причини відсутні відповідні оголошення у розділі заголовка. Подібно до оголошення HTML-тегів або змінних JavaScript для програмування браузера, локальне сховище може бути використане безпосередньо з моменту початку виконання скрипту.

2.4. Діаграма пакетів та взаємозв'язки компонентів

На рисунку 2.3 представлена діаграма пакетів, що ілюструє взаємозв'язок між програмним кодом та візуальними активами проєкту. Точкою входу в програму слугують HTML-файли, кожен з яких відповідає окремому ігровому рівню.

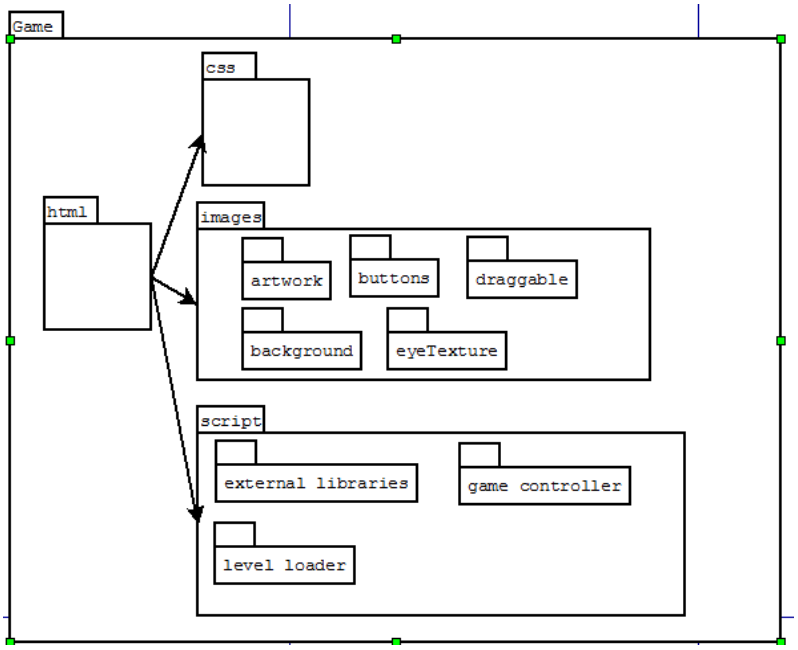

Рисунок 2.3 - Діаграма пакетів

HTML-компоненти залежать від таких елементів:

- CSS-файли забезпечують структурування та стилізацію елементів інтерфейсу.

- Файли зображень надають необхідні візуальні активи. Зверніть увагу, що папка зображень організована в окремі підрозділи, що відповідають різним типам завантажуваних зображень.

- JavaScript-файли відповідають за всю функціональність програми.

Пакет JavaScript включає файли, критично важливі для виконання гри, такі як скрипти ігрових рівнів та функціональність, що уможливлює їхню роботу. Детальніше про ці аспекти буде викладено пізніше в роботі. Крім того, пакет JavaScript містить зовнішні бібліотеки, необхідні для функціонування модуля WebGL.

Хоча це не відображено на рисунку 2.3 через відсутність прямого виклику зовнішніх бібліотек або ресурсів, локальне сховище є життєво важливим компонентом, що забезпечує запуск та функціонування гри. Це включає збереження файлу gameloop, який знаходиться у пакеті контролера гри. Питання локального сховища буде докладніше описано у відповідному розділі.

2.5. Схема програмних компонентів

Цей розділ надає огляд ключових програмних компонентів, необхідних для функціонування гри. Структура представлення матеріалу спрямована на забезпечення розуміння індивідуального функціонування кожного компонента, оскільки детальне вивчення їх взаємодій могло б ускладнити сприйняття. Наприклад, об'єкт ігрового циклу ініціалізується однаково з використанням JQuery та локального сховища і викликається постійно протягом гри. Розгляд його різноманітного використання у різні моменти часу вимагав би постійного перемикання між концепціями. Тим не менш, для

загального розуміння послідовності виконання гри, на наступній сторінці включено схему робочого процесу.

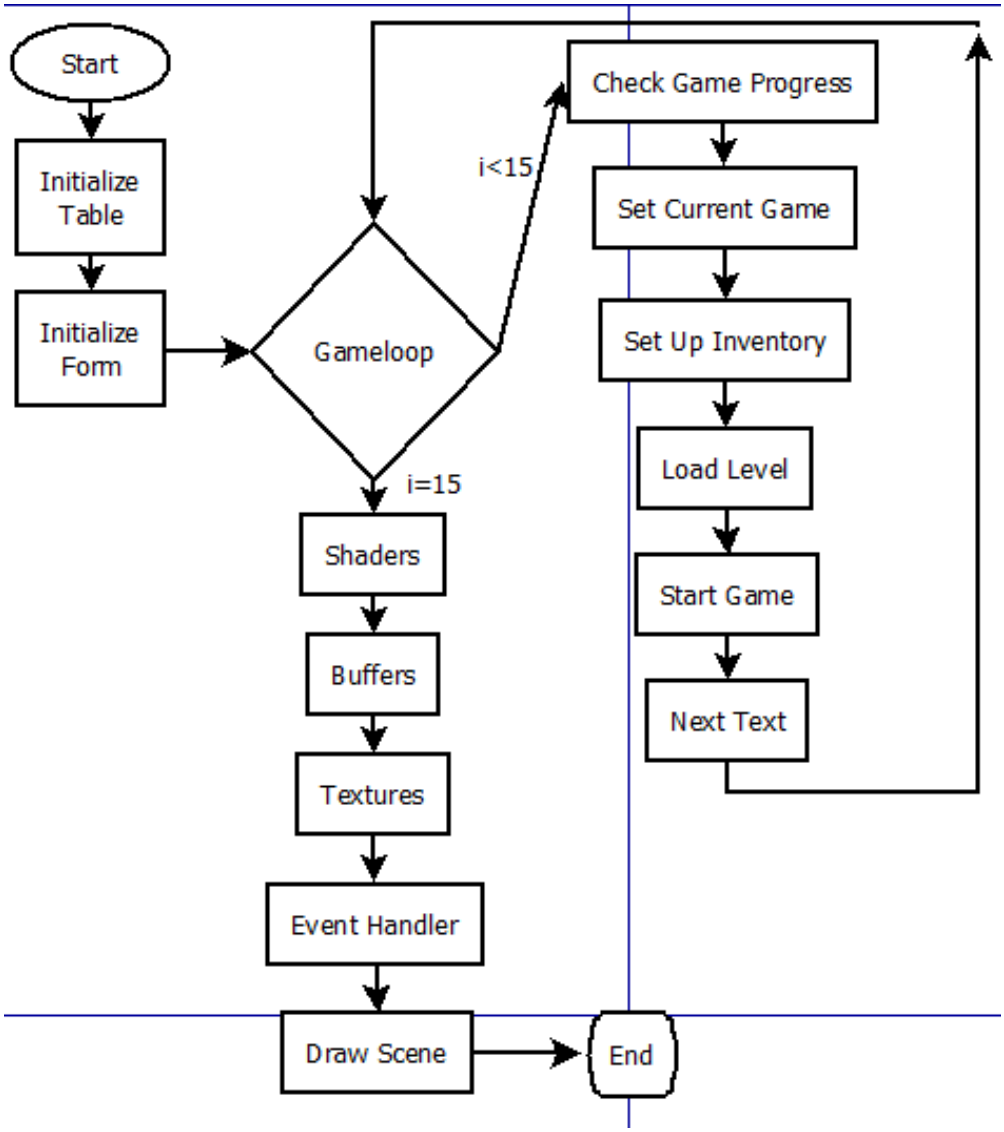

Рисунок 2.4 - Схема робочого процесу

Як показано на рисунку 2.4, після початку гри наступним етапом є створення ігрового облікового запису. Цей процес включає ініціалізацію таблиці для відображення існуючих ігрових облікових записів та форми, що дозволяє користувачам створити новий обліковий запис. Після вибору створеного облікового запису для використання, користувач починає гру, і в цей момент активується ігровий цикл.

Протягом ігрового процесу кожна сторінка проходить через цикл, у якому система перевіряє прогрес користувача. Поточний рівень та будь-який збережений інвентар автоматично зберігаються в обліковому записі користувача. Після завершення етапу управління обліковими записами починається функціональність індивідуальних рівнів. Спочатку необхідні елементи створюються за допомогою завантаження рівня, а взаємодія для цих рівнів визначається у функції start game. Подальші дії після виконання конкретного ігрового завдання рівня вказуються у функції next text, яка потім переводить користувача на новий рівень. Цей цикл продовжується, поки користувач проходить через різні ігрові рівні.

Цикл завершується, коли користувач проходить усі рівні; після цього він може перейти до модуля WebGL. Цей заключний рівень вимагає низки послідовних кроків, починаючи з ініціалізації шейдерів та буферів, подальшої обробки подій миші, завантаження текстур та завершуючи відображенням сцени.

2.6. Процедури рендерингу 3D-сцени у WebGL

У динамічних віртуальних середовищах необхідний постійний цикл оновлення для відображення поточного стану сцени. Цей цикл безперервно перевіряє зміни та ініціює рендеринг поточного кадру. Після завершення всіх обчислень для кадру, сцена переходить до етапу промальовування.

```
function tick() {
    requestAnimFrame(tick);
    drawScene();
}
```

2.6.1. Налаштування вікна перегляду та перспективи

Приступаючи до рендерингу сцени, першочерговим завданням є визначення вікна перегляду (viewport) — області на canvas, куди буде

здійснюватися виведення зображення. Ця операція включає встановлення координат нижнього лівого кута та розмірів вікна перегляду. Після цього буфери кольору та глибини очищаються, готуючи їх до нового кадру. Далі налаштовується проекційна матриця (pMatrix), яка визначає перспективне спотворення сцени, включаючи кут огляду (в даному випадку 45 градусів), співвідношення сторін, а також ближню та дальню площини відсікання.

```
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
```

2.6.3. Конфігурація освітлення сцени

Наступним кроком є встановлення параметрів освітлення, що впливають на тривимірний світ. Інформація про освітлення може динамічно надходити з HTML-елементів введення, дозволяючи користувачеві змінювати ці параметри. У даному випадку передбачається наявність двох типів освітлення: направленого (directional light) та навколишнього (ambient light).

Напрям освітлення та його колір (R, G, B компоненти) визначаються окремо. Для візуального ефекту райдужки ока, освітлення трохи зміщене в червоний спектр. Це надає білій райдужці легкого червонуватого відтінку, але при цьому не настільки інтенсивного, щоб замаскувати текстуровані капіляри ока. Значення, отримані з HTML-інтерфейсу, передаються до шейдерних програм WebGL:

```
<h2>Directional light:</h2>
<td><b>Direction:</b><td>X: <input type="text" id="lightDirectionX" value="-1.0" />
<td><b>Color:</b><td>R: <input type="text" id="directionalR" value="0.8" /><td>G:

<h2>Ambient light:</h2>
<td><b>Color:</b><td>R: <input type="text" id="ambientR" value="0.2" /><td>G: <inpu
```

Передача цих значень до шейдерів здійснюється таким чином:

```
gl.uniform3f(
    shaderProgram.ambientColorUniform,
    parseFloat(document.getElementById("ambientR").value),
    parseFloat(document.getElementById("ambientG").value).
    parseFloat(document.getElementById("ambientB").value)
);

var lightingDirection = [
    parseFloat(document.getElementById("lightDirectionX").value),
    parseFloat(document.getElementById("lightDirectionY").value),
    parseFloat(document.getElementById("lightDirectionZ").value)
];

var adjustedLD = vec3.create();
vec3.normalize(lightingDirection, adjustedLD);
vec3.scale(adjustedLD, -1); // Інвертування напрямку для джерела світла
gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);

gl.uniform3f(
    shaderProgram.directionalColorUniform,
    parseFloat(document.getElementById("directionalR").value),
    parseFloat(document.getElementById("directionalG").value),
    parseFloat(document.getElementById("directionalB").value)
);
```

Після налаштування освітлення, модельно-видова матриця (mvMatrix) ініціалізується як одинична матриця. Це забезпечує початкове, "чисте" положення об'єкта. Далі камера віртуально переміщується на шість одиниць назад вздовж осі Z, щоб забезпечити адекватний огляд сцени перед початком малювання. Потім ця матриця множиться на eyeRotationMatrix, яка містить дані про обертання ока, збережені у буфері.

```
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [0, 0, -6]);
mat4.multiply(mvMatrix, eyeRotationMatrix);
```

Завершальним етапом є безпосереднє промальовування трикутників, які формують тривимірну сферу. Це досягається шляхом прив'язки до буфера індексів вершин сфери (eyeVertexIndexBuffer), встановлення уніформ матриць (включаючи mvMatrix та pMatrix) у шейдерних програмах, і,

нарешті, виклику функції gl.drawElements, яка рендерить елементи (трикутники) на основі даних з прив'язаних буферів.

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, eyeVertexIndexBuffer);
setMatrixUniforms();
gl.drawElements(gl.TRIANGLES, eyeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0)
```

Ці послідовні кроки дозволяють створювати динамічні та візуально інтерактивні 3D-сцени в середовищі WebGL.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ СТРАТЕГІЇ ІГРОВОГО ДОДАТКУ

3.1. Реалізація програмного компоненту gameLoop у JavaScript

JavaScript як мова програмування демонструє високу гнучкість щодо оголошення типів даних, дозволяючи використовувати єдине ключове слово var для ініціалізації цілих чисел, символів, рядків, масивів та об'єктів. Ця особливість є особливо цінною для реалізації об'єкта gameLoop. Основне призначення об'єкта gameLoop полягає у відстеженні прогресу користувача у грі. Для цього використовується метод checkGameProgress(), який функціонує як регулятор, спрямовуючи гравців на відповідні рівні.

Незважаючи на те, що JavaScript не є внутрішньо об'єктно-орієнтованою мовою, архітектура об'єкта gameLoop імітує принципи об'єктно-орієнтованого програмування за допомогою створення змінних-членів та методів-полів.

3.1.1. Перевірка прогресу гри (checkGameProgress())

Для ефективного відстеження прогресу гри необхідно ідентифікувати поточного гравця. Це досягається викликом функції setCurrentPlayer(), яка отримує дані поточного гравця з локального сховища та зберігає їх у полі currentPlayer об'єкта gameLoop. Детальний опис механізму локального сховища буде надано у відповідному розділі. Важливо зазначити, що поле currentPlayer, яке спочатку ініціалізується порожнім рядком (currentPlayer: ""), саме по собі є об'єктом. Цей об'єкт містить ідентифікаційну інформацію (наприклад, ім'я гравця), дані про ігровий прогрес (поточний рівень, виконані сюжетні маркери) та інформацію про інвентар гравця. Після збереження об'єкта гравця більшість операцій полягатимуть у читанні інформації з цього об'єкта, оскільки запис відбувається лише при значних змінах у прогресі.

3.1.2. Встановлення поточного рівня

Першим значущим завданням є отримання поточного рівня гравця:

```
var currentLevel = gameLoop.currentPlayer.level;
```

Це числове значення використовується для ітерації по карті прогресії гри, що зберігає відносні шляхи до різних рівнів. Карта прогресії зберігається як поле об'єкта gameLoop:

```
gameProgressionMap: ["accountManager.html", "storyScreen.html", "hallway.html", "hallway2
```

Для формування абсолютного шляху до поточного рівня отримується значення URL хоста (наприклад, base: "http://www.al.com/TheiProject/"). Рядок сервера конкатенується з відносним шляхом для формування повного URL поточного рівня:

```
var currentLevelURL = gameLoop.base + gameLoop.gameProgressionMap[currentLevel];
```

Таким чином, визначається очікуване місцезнаходження гравця. Наприклад, якщо гравець щойно розпочав гру після створення облікового запису, він перебуватиме на першому рівні та буде перенаправлений на початковий екран історії: http://www.ald.com/TheiProject/storyscreen.html.

У випадку, якщо поточний URL користувача не відповідає очікуваному currentLevelURL, система перевіряє його позицію. Якщо користувач перебуває на попередньому рівні, йому дозволяється залишатися, оскільки можливим є повернення до попередніх веб-сторінок. Однак, якщо користувач намагається перейти на вищий рівень не за порядком, це розцінюється як спроба пропустити рівні. У такому випадку, під час ітерації по карті прогресії, якщо поточний URL відповідає певному рівню, і його ідентифікатор перевищує рівень поточного гравця, відбувається перенаправлення гравця на рівень, на якому він повинен перебувати:

```
if (document.URL != currentLevelURL) {
    for (var i = 0; i < gameLoop.gameProgressionMap.length; i++) {
        currentLevelURL = gameLoop.base + gameLoop.gameProgressionMap[i];
        if (document.URL == currentLevelURL) {
            if (i > currentLevel) {
                window.location.replace(gameLoop.gameProgressionMap[currentLevel]);
            }
        }
        break; // Цей 'break' потребує уваги, оскільки він зупиняє цикл після першої перевірки
    }
}
```

3.1.3. Налаштування інвентарю (setUpInventory())

Після підтвердження коректного рівня гравця відбувається ініціалізація інвентарю за допомогою виклику gameLoop.setUpInventory(). Завантаження інвентарю не відбувається миттєво при вході користувача на рівень; замість цього, до кнопки інвентарю прив'язується обробник подій click, що дозволяє користувачеві викликати інвентар за бажанням:

```
$(".inventoryButton").click(function() {
```

Для забезпечення наявності лише одного екземпляра інвентарю в певний момент, здійснюється перевірка на його попереднє створення:

```
if (gameLoop.inventoryNotYetCreated) {
```

Якщо інвентар ще не створений, флаг gameLoop.inventoryNotYetCreated встановлюється на false, після чого відбувається створення інвентарю. Оскільки елементи додаються до DOM, який має деревоподібну структуру, необхідно знайти батьківський вузол для приєднання нового дочірнього вузла. У даному випадку кнопка інвентарю обирається як батьківський елемент. Новий дочірній вузол, що має ідентифікатор "inventory", включає шлях до зображення, призначеного для візуального представлення інвентарю. З художньої точки зору, використання зображення є бажанішим, ніж створення поля за допомогою CSS:

```
$('.inventoryButton').append('<p id="inventory"><img src="images/draggable/InventoryFinal
```

Оскільки це інвентар, користувачам надається можливість перетягувати його по екрану, що дозволяє розміщувати його таким чином, аби він не заважав вибору інших елементів. Крім того, інвентар є скидуваним (droppable), оскільки в нього будуть розміщуватися елементи. Ця функціональність реалізується за допомогою JQuery UI:

```
$("#inventory").draggable();
$("#inventory").droppable({
```

У контексті використання, перетягування інвентарю є інтуїтивно зрозумілим: користувач вибирає елемент і розміщує його в бажаному місці. Проте, обробка скидання на інвентар є складнішою. Для визначення того, що скидається на інвентар і чи слід це приймати, використовується атрибут scope. Наприклад, перетягуваному елементу може бути присвоєно специфічне ім'я, наприклад "important", щоб позначити його як скидуваний, а потім відповідний ідентифікатор може бути призначений скидуваному елементу для позначення типу елемента, який приймається. Це гарантує, що приймаються лише ті елементи, які можуть бути збережені, а не будь-який перетягуваний елемент:

```
scope: "sortable",
```

Після того, як інвентар отримав область, можна визначити дії, які виконуються при легітимному скиданні, що здійснюється за допомогою функції зворотного виклику drop:

```
drop: function(event, ui) {
```

У цій функції спочатку отримується ідентифікатор перетягуваного елемента, який був скинутий:

```
var droppedItemId = ui.draggable.attr("id");
```

Знаючи ідентифікатор скинутого елемента, необхідно визначити, чи є цей елемент першим, який заповнює інвентар, або він замінює вже збережений елемент. У першому випадку перевіряється відсутність елемента в інвентарі, і якщо ця умова істинна, скинутий елемент зберігається в інвентарі:

```
if (!gameLoop.currentPlayer.inventory) {
    gameLoop.storeInventory(droppedItemId);
}
```

Зберігання інвентарю вимагає двох записів: один в об'єкт у пам'яті, а інший у сховище. Запис у сховище пояснюється в розділі про локальне сховище, тоді як запис у пам'ять вимагає лише збереження переданого ідентифікатора:

```
gameLoop.currentPlayer.inventory = itemID;
```

Щоб уникнути дублювання елемента (один скинутий на інвентар зі сторінки, а інший завантажений з інвентарю), дозволяється існувати лише тому, який завантажений з інвентарю. Таким чином, скинутий елемент видаляється:

```
$(ui.draggable).remove();
```

Потім завантажується елемент з інвентарю:

```
gameLoop.loadInventory();
```

Завантаження з інвентарю технічно передбачає повторне створення елемента. При збереженні елемента зберігається лише його ідентифікатор, який потім використовується для створення кінцевого продукту.

Після ініціалізації функції скидання відбувається завантаження інвентарю. Для цього спочатку перевіряється, чи вже існує елемент, який необхідно завантажити, на сторінці:

```
if (gameLoop.checkIfItemExists()) {
```

Список можливих для збереження елементів є невеликим, тому визначено масив з їхніми можливостями:

```
var possibleDraggableItems = ["chair", "mechSpider", "wrench", "trinket", "falseTrinket1'
```

Здійснюється ітерація по цьому списку, і перевіряється наявність будь-якого з елементів на сторінці за допомогою виклику функції JavaScript getElementById. Якщо елемент існує, функція повертає true, і повідомлення передається у власну функцію checkIfItemExists:

```
for (var i = 0; i < possibleDraggableItems.length; i++) {
    if (document.getElementById(possibleDraggableItems[i])) {
```

Після того, як ця умова стає істинною, результат зберігається в об'єкті gameLoop для легкого доступу:

JavaScript

```
gameLoop.duplicateItemonPage = true;
```

Визначення наявності дубліката на сторінці є важливим, оскільки функціональність JQuery працюватиме лише з одним елементом з даним ідентифікатором; дублікат елемента просто існував би на сторінці, не виконуючи жодних дій. Для розрізнення дублікатів одному буде присвоєно стандартне ім'я (наприклад, "chair"), а копіям – розширення (наприклад, "chaircopy"). Після цього можна продовжити завантаження інвентарю.

У попередній функції, під час перевірки на дублікат, поле duplicateItemonPage встановлювалося на true, якщо дублікат вже був

збереженим елементом на сторінці. Це допомагає, якщо виявляється, що елемент, який необхідно завантажити, вже існує. Якщо це не так, до імені елемента додавався б порожній рядок, залишаючи ім'я незмінним:

```
var draggableNameAddition = "";
```

Проте, якщо б була копія, до неї додавався б рядок "copy".

```
if (gameLoop.duplicateItemonPage) {
    draggableNameAddition = "copy";
}
```

Оскільки до імені елемента додається суфікс, можливо, що завантажуваний елемент вже був копією (або через те, що копія була збережена раніше, коли користувач виконував дії на іншій сторінці, або через послідовність збережень на поточній сторінці). Тому, коли отримується ім'я елемента, збереженого в інвентарі, перевіряється наявність суфікса копії за допомогою регулярного виразу "copy". Якщо це копія, рядок "copy" відокремлюється на цьому етапі і буде доданий пізніше, після точного створення елемента. Причина полягає в тому, що розглянуті випадки перевірятимуть лише кореневе ім'я елемента і створюватимуть саме цей елемент. Тобто, "chair" і "chaircopy" створять один і той же елемент; вони просто матимуть різні імена. Наприклад, для завантаження стільця:

```
case "chair":
    $(".inventoryButton").append('<p id="chair' + draggableNameAddition + '"><img src="im
    $("#chair" + draggableNameAddition).draggable({
        scope: "important"
    });
    break;
```

Зверніть увагу, що коли до стільця додався суфікс, якщо стілець був оригінальним, до нього додавався порожній рядок, що призводило до "chair", а якщо це була копія, додавалося "copy", що призводило до "chaircopy". Всі інші елементи створюються аналогічним чином, і кожному надається

відповідна область. Стільцю надається область "important", оскільки він запускає подію історії. Однак елементам, які не запускають події історії, таким як "falsetrinket", не буде надано області:

```
case "falseTrinket1":
    $(".inventoryButton").append('<p id="falseTrinket1' + draggableNameAddition + '"><img
    $("#falseTrinket1" + draggableNameAddition).draggable();
    break;
```

Останнє використання JQuery в об'єкті gameLoop відбувається, коли встановлюється рівень гравця після того, як він пройшов через двері, які були відчинені після завершення сюжетної події. Візьмемо приклад другого коридору, де гравцям буде дозволено пройти через двері лише після першої зустрічі з "i". Перевіряється ця умова, і якщо вона істинна, створюється зображення відчинених дверей, які функціонують як гіперпосилання. Коли користувач натискає на посилання, його рівень встановлюється на запропонований рівень за допомогою виклику функції setLevel:

```
checkFirstEncounter: function() {
    if (gameLoop.currentPlayer.storyProgressMarker == "iMet") {
        $('.frontDoor').html('<a href="javascript:gameLoop.setLevel(5)" style="text-decor
    }
}
```

Функція setLevel працює шляхом перевірки рівня гравця, як зазначено в об'єкті gameLoop у пам'яті. Якщо збережений рівень нижчий, гравець, ймовірно, не просунувся далі, тому рівень встановлюється на вищий (у цьому випадку на п'ятий). Якщо рівень гравця вищий або рівний, це означає, що гравець вже просунувся далі в грі, тому записаний вищий рівень зберігається без перезапису:

```
if (levelNumber > gameLoop.currentPlayer.level) {
    gameLoop.currentPlayer.level = levelNumber;
}
window.location.replace(gameLoop.gameProgressionMap[levelNumber]);
```

3.2. Програмний компонент: Об'єкт рівня (Level Object)

Крім своєї корисності у створенні об'єкта gameLoop для відстеження прогресу користувача, бібліотека JQuery також активно застосовується у реалізації логіки ігрових рівнів, забезпечуючи динамічне оновлення сторінки у відповідь на взаємодію користувача. Початковим етапом є завантаження рівня.

3.2.1. Завантаження рівня (loadLevel())

У HTML-структурі елементам сторінки надається мінімальна кількість даних, оскільки основна інформація передається через змінні JavaScript. У функції loadLevel() елементи завантажуються на веб-сторінку наступним чином:

```
loadLevel: function() {
    var level = '<div id="chair" class="ui-widget-content"><img src="images/draggable/sta
    $(".thisRoom").html(level);
}
```

3.2.2. Початок гри (startGame())

На цьому етапі розробки елементи є лише статичними зображеннями. Для забезпечення їх інтерактивності викликається функція startGame():

```
levelOne.startGame();
```

У цій функції стілець робиться перетягуваним (draggable), що дозволяє користувачеві переміщувати його у визначене місце на сторінці. Для цього оголошується зона скидання (droppable). Ця зона має таку ж область видимості ("important") і містить функцію drop, яка активується, коли користувач скидає правильний об'єкт. У контексті першого рівня, користувачеві пропонується перемістити стілець у певне місце для доступу до певної частини рівня. Коли стілець скидається, активується традиційна

2D-сцена діалогу, що складається з зображень двох персонажів, які розмовляють, та текстового поля:

```
$("#droppable").droppable({
    scope: "important",
    drop: function(event, ui) {
        $('.thei').html('<img src="images/artwork/theiTotemFinal.png" style="border-style
        $('.atticus').html('<img src="images/artwork/atticusFinal.png" style="border-styl
        $('.textbox').html('<img src="images/background/textBackground.jpg" style="border
        $('#backButton').html('<img src="images/buttons/FinalBackButton.jpg" style="borde
        $('#nextButton').html('<img src="images/buttons/FinalNextButton.jpg" style="borde
```

Текст завантажується з відповідного поля об'єкта рівня:

```
$('.textboxText').html(levelOne.firstText);
```

У цьому випадку викликається текст:

```
firstText: "<h3>i</h3><p>Are you... organic?</p>",
```

3.2.3. Наступний текст (nextText())

Після відображення початкового тексту викликається функція nextText(), яка прив'язує функціонал кліку до кнопок "назад" (back) та "далі" (next), дозволяючи користувачеві прокручувати послідовність діалогів:

```
levelOne.nextText();
```

Принцип роботи функції nextText() полягає у збільшенні значення поля index (яке є полем об'єкта першого рівня) щоразу, коли користувач натискає кнопку "далі". Це призводить до перезапису HTML-вмісту JQuery-об'єкта textboxText. Коли гравець прокрутив увесь текст, він завершує першу зустріч з "i" і може перейти до наступного завдання. Отже, маркер прогресу історії оновлюється, і користувач перенаправляється на наступну сторінку, яка у даному випадку є коридором, що вів до першого рівня гри. Раніше у цьому

коридорі були зачинені двері, але тепер, коли користувач зустрів "і", двері будуть відкриті:

```
$("#nextButton").click(function() {
    levelOne.index++;
    switch (levelOne.index) {
        case 2:
            $('.textboxText').html(levelOne.secondText);
            break;
        // ... інші випадки
        case 5:
            window.gameLoop.currentPlayer.storyProgressMarker = "iMet";
            window.location.replace("hallway2.html");
            break;
    }
});
```

Поле index спочатку встановлюється на одиницю і збільшується до п'яти у цьому ігровому рівні, причому кожен пронумерований випадок представляє діалоговий текст. Випадок чотири є останнім текстовим випадком, а випадок п'ять перенаправляє користувача. Тобто, випадок один надає перший текст, випадок два — другий текст і так далі. Щоб забезпечити ідентичну функціональність для кнопки "назад", як і для кнопки "далі", випадки від одного до чотирьох призначаються відповідним текстовим блокам.

Тому у функції кнопки "назад" передбачається, що користувач може досягти випадку нуль, і коли це відбувається, індекс автоматично встановлюється назад на одиницю, залишаючи користувача з текстом випадку один — першим текстом:

```
$("#backButton").click(function() {
    levelOne.index--;
    switch (levelOne.index) {
        case 0:
            levelOne.index = 1;
            break;
        case 1:
            $('.textboxText').html(levelOne.firstText);
            break;
    }
});
```

Більшість ігрових рівнів мають подібну структуру, з деякими незначними відмінностями, необхідними для специфічної функціональності. Наприклад, у game.js користувач скидає перетягуваний елемент у правильне місце та отримує статичний текст, тоді як у game2.js потрібен невеликий анімаційний ефект. Користувачеві необхідно перемістити механічного павука, і коли елемент скидається у правильне місце, він повинен "ожити" та "вискочити" на користувача. Для реалізації цього завдання використовуються функції JQuery animate, fadeOut та hide. Механічний павук призначений для перетягування, тому, щоб він був нерухомим і рухався детерміновано, оригінальний павук видаляється і замінюється новим (лише для розрізнення). Потім цей новий павук анімується, вказуючи, що він повинен рухатися на двісті п'ятдесят пікселів ліворуч:

```
drop: function(event, ui) {
    $("#mechSpider").hide();
    $(".critter").html('<img src="images/artwork/fivelegged.png" style="position:absolute
    $(".critter").animate({left:'250px'}, function() {
```

Після завершення руху функція animate викликає іншу функцію. До цього моменту користувач спостерігав, як павук рухається по екрану. Щоб змусити його "вискочити" на користувача, рухомий павук видаляється і замінюється більшим зображенням, що вказує на наближення павука. Передбачається, що павук прикріпиться до голови персонажа і дозволить гравцеві спілкуватися з "і" через екран на череві механічного павука. Таким чином, зображення великого павука має повільно зникати, переходячи у діалоговий обмін, подібний до того, що є у game.js:

```
$(".critter").hide();
$(".biggerCritter").html('<img src="images/artwork/televisionWithLegs.png" style="positic
$(".biggerCritter").fadeOut("slow", function() {
```

Наступний рівень гри, game3.js, використовує ті ж принципи, що й перші два ігрові рівні, проте четвертий рівень гри (game4.js) вводить новий

ігровий елемент. До цього моменту оголошення області скидання було лише формальністю, оскільки кожен ігровий рівень містив лише один перетягуваний і один скидуваний елемент. Однак у game4.js користувач обирає з трьох перетягуваних елементів, з яких лише один буде прийнятий скидуваним. Два "хибних" скидуваних елементи не отримують область "important", тому вони не будуть прийняті зоною скидання:

```
$("#trinket").draggable({
    scope: "important"
});
$("#falseTrinket1").draggable();
$("#falseTrinket2").draggable();
$("#droppable").droppable({
    scope: "important",
});
```

Ігрове завдання на останньому рівні гри відрізняється від завдань перших чотирьох рівнів. Тут відсутня функціональність перетягування чи скидання; натомість, передбачається введення коду, що полегшується JQuery. На четвертому рівні гри гравець дізнається пам'ятну дату для одного з NPC, і ця дата може бути використана як пароль, який потрібно ввести на цифровій клавіатурі на цьому рівні. Кожна цифра є об'єктом JQuery, якому присвоєна подія кліку. Коли користувач натискає на цифру, то вона додається до змінної пароля. Після введення правильного коду користувач побачить наступну сцену діалогу. Приклад цифри:

```
$(".one").click(function() {
    levelFive.password += '1';
    if (levelFive.password == '0214') {
        levelFive.loadText();
    }
});
```

3.3. Збереження стану гри за допомогою локального сховища

Локальне сховище відіграє критично важливу роль у збереженні та відновленні стану гри, а також у відстеженні поточного прогресу

користувача. Оскільки HTML за своєю природою є безстатевим протоколом, при переході між веб-сторінками (наприклад, з accountManager.html на першу ігрову сторінку) відбувається втрата всієї інформації про попередні зміни. Для забезпечення безперервності ігрового процесу та збереження відчуття сталості, необхідно перезавантажувати всю важливу збережену інформацію з локального сховища.

3.3.1. Механізм завантаження поточного ігрового стану (setCurrentGame())

Першим елементом даних, який необхідно завантажити, є інформація про поточний активний ігровий запис. Це досягається шляхом ітерації по збережених записах у локальному сховищі, їх парсингу та ідентифікації того запису, для якого встановлено значення currentPlayer як true. Для зручності подальшого доступу цей запис зберігається в пам'яті як член об'єкта gameLoop:

```
setCurrentGame: function() {
    for (var key in localStorage) {
        if (/Player:\d+/.test(key)) {
            var parsedPlayer = JSON.parse(window.localStorage.getItem(key));
            if (parsedPlayer.currentGame) {
                gameLoop.currentPlayer = parsedPlayer;
            }
        }
    }
}
```

Після завантаження ігрового запису в пам'ять, стає можливим перевірити поточний рівень гравця, ідентифікувати завершені сюжетні події та визначити вміст інвентарю гравця.

3.3.2. Управління прогресом рівня

Більшість операцій, пов'язаних з рівнем, обробляються JavaScript після того, як дані рівня завантажені в пам'ять. Цей підхід є ефективним до

моменту, коли користувач переходить на нову сторінку або відбувається значуща ігрова подія. У таких випадках задіюється локальне сховище. Наприклад, при переході на нову сторінку викликається функція setLevel(), яка приймає поточний номер рівня як аргумент і порівнює його зі збереженим рівнем. Оскільки ця функція викликається лише під час легітимних переходів, система вважає, що користувача має бути перенаправлено на вищий рівень. Відповідно, якщо поточний рівень вищий, збережений рівень оновлюється до цього вищого значення. Цей новий рівень зберігається у сховищі, після чого користувач перенаправляється на відповідну нову сторінку:

```
setLevel: function(levelNumber) {
    if (levelNumber > gameLoop.currentPlayer.level) {
        gameLoop.currentPlayer.level = levelNumber;
    }
    window.localStorage.setItem("Player:" + gameLoop.currentPlayer.id, JSON.stringify(gam
    window.location.replace(gameLoop.gameProgressionMap[levelNumber]);
}
```

3.3.3. Збереження інвентарю (storeInventory())

Окрім збереження прогресу рівня гравця, ще одним важливим завданням локального сховища є збереження інвентарю гравця. Це може відбуватися у двох сценаріях: коли інвентар порожній (оскільки він ніколи не використовувався) або коли у сховищі вже є предмет, і користувач вирішує замінити старий предмет. В обох випадках викликається функція storeInventory() з ідентифікатором предмета, який необхідно зберегти. Цей ідентифікатор спочатку присвоюється змінній JavaScript, а потім зберігається у локальному сховищі:

```
storeInventory: function(itemID) {
    gameLoop.currentPlayer.inventory = itemID;
    window.localStorage.setItem("Player:" + gameLoop.currentPlayer.id, JSON.stringify(gam
    gameLoop.idItemLoadedToInventory = itemID;
}
```

3.3.4. Перевірка прогресу гри (checkFirstEncounter())

Останнім важливим обов'язком локального сховища є перевірка виконання гравцями певних ігрових завдань. Наприклад, гравцям не буде дозволено пройти за другий коридор, поки вони не зустрінуться з "i". Ця перевірка здійснюється у gameLoop і використовує об'єкт запису, збережений в об'єкті gameLoop:

```
checkFirstEncounter: function() {
    if (gameLoop.currentPlayer.storyProgressMarker == "iMet") {
        $('.frontDoor').html('<a href="javascript:gameLoop.setLevel(5)" style="text-decor
    }
}
```

Двері залишатимуться зачиненими, поки гравець не зустрінеться з "i". Після цього, коли значення iMet буде true, гравець побачить відкриті двері. Значення iMet встановлюється, коли гравці завершують свою першу зустріч з "i" у файлі game.html:

```
case 5:
    window.gameLoop.currentPlayer.storyProgressMarker = "iMet";
    window.localStorage.setItem("Player: " + window.gameLoop.currentPlayer.id, JSON.strir
    window.location.replace("hallway2.html");
    break;
```

Цей п'ятий випадок представляє останню діалогову опцію з "i", після чого користувач перенаправляється до коридору, де раніше були зачинені двері, які тепер виявляються відчиненими.

3.4. Інтеграція WebGL для рендерингу 3D-графіки у веб-середовищі

3.4.1. Концептуальні основи WebGL

WebGL функціонує на основі тих самих стандартів, що й традиційний OpenGL, надаючи доступ до програмованого графічного конвеєра через шейдерні програми. Ключова відмінність полягає в тому, що WebGL надає

графічні операції для JavaScript, що виконується у браузері, тоді як традиційний OpenGL забезпечує ці операції для програм, що безпосередньо взаємодіють з бібліотеками OpenGL операційної системи. JavaScript використовує операції WebGL для рендерінгу сцен за допомогою елементів HTML Canvas.

Для рендерінгу сцени необхідно виконати низку послідовних завдань, включаючи ініціалізацію canvas, шейдерів, буферів, обробників подій та текстур. Після цього сцена малюється у циклі, рендерячи кожен кадр послідовно. Для створення рухомого очного яблука, згаданого у попередніх розділах, необхідно дотримуватися кожного з цих кроків. Завдання реалізується шляхом створення тривимірної сфери, її текстурування, налаштування освітлення та реагування на вхід користувача. Кожен крок виконується відповідними функціями у межах функції webGLStart():

```
function webGLStart(){
    var canvas = document.getElementById("iWorld");
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    canvas.onmousedown = handleMouseDown;
    canvas.onmousemove = handleMouseMove;
    tick();
}
```

3.4.2. Ініціалізація Canvas

Графіка WebGL будується на основі HTML Canvas елементів. Тому першим кроком у JavaScript є отримання посилання на цей елемент. Елемент Canvas, разом з його розмірами, визначається у тілі HTML:

```
<canvas id="iWorld" style="border: none;" width="500" height="500">
```

Оскільки елемент є частиною DOM (Document Object Model), посилання на нього може бути отримане JavaScript та збережене у змінній

```
var canvas = document.getElementById("iWorld");
```

Після отримання посилання на елемент Canvas, наступним кроком є отримання посилання на його асоційований OpenGL контекст:

```
gl = canvas.getContext("experimental-webgl");
```

На момент написання даного тексту, реалізація WebGL для елемента Canvas підтримується в останніх версіях всіх браузерів. Перед продовженням необхідно перевірити існування контексту OpenGL:

```
if (!gl){
    alert("Could not initialise WebGL, sorry :-(");
}
```

За умови, що контекст OpenGL існує, наступним кроком є встановлення розмірів вікна перегляду (viewport), яке він буде рендерити. Ці розміри отримуються з атрибутів width та height HTML Canvas елемента:

```
gl.viewportWidth = canvas.width;
gl.viewportHeight = canvas.height;
```

3.5. Методи створення художніх активів за допомогою GIMP

Створення візуальних активів для гри, за відсутності можливості найняти професійного художника, здійснювалося шляхом модифікації наявних фотографій з метою імітації ручної роботи. Важливою передумовою було суворе дотримання законів про авторське право: навіть якщо зображення значно модифіковано, воно залишається під захистом авторського права. Отже, всі зображення або створювалися програмістом, або отримувалися за безоплатною (royalty-free) ліцензією.

3.5.1. Виявлення країв (Edge Detection)

Після отримання відповідного зображення застосовувався художній фільтр, який емулює ручне малювання. Особливо ефективним виявився фільтр "виявлення країв" (edge detection). Цей фільтр ідентифікує та підкреслює будь-які краї на зображенні, одночасно приглушуючи нечіткі деталі. Метою є виділення контурів об'єктів, подібно до того, як це відбувається при ручному малюванні. У реальному світі краї представляють контури об'єктів і зазвичай є чорними. Однак, щоб алгоритм зменшив деталі, він робить їх чорними, а для підкреслення країв змінює їх колір, роблячи будь-які чорні краї білими.

У зображенні, складеному з пікселів, чорний піксель не завжди означає край, оскільки смуга чорних пікселів може формувати чорний об'єкт. Щоб визначити, чи є щось краєм, алгоритм аналізує будь-яку зміну інтенсивності кольору. Якщо між сусідніми пікселями спостерігається невеликий градієнт, існує висока ймовірність, що це той самий матеріал на зображенні. Проте, коли виявляється різка зміна кольору між пікселями (наприклад, з яскраво-червоного на чорний), ймовірно, присутній кордон.

Цю варіацію можна представити математично. Пікселі зображення можуть бути згруповані у рядки та стовпці. Якщо розглядати кожен рядок як лінійне рівняння, де вісь x представляє положення пікселя, а вісь y — інтенсивність кольору в цьому положенні, можна побудувати графік рівняння. Коли на графіку є різкі зміни, шляхом знаходження першої похідної можна визначити моменти швидкої зміни нахилу, що вказує на різку зміну варіації кольору. Ці інтервали, де присутній край, будуть позначені товстою білою лінією.

Для більш тонких градієнтів потрібен інший алгоритм. Розглянемо, наприклад, фотографію неба з хмарами. Коли зображення переходить від світлого неба до світлих країв хмар і далі до темно-сірого ядра хмар, небажано, щоб товсті краї позначали ці переходи. Необхідні тонші краї, які

позначають більш тонкий перехід, а не різку зміну, позначену першою похідною. Демаркація тонкої зміни здійснюється за допомогою другої похідної.

Рисунок 3.1 - Фото до обробки країв

Рисунок 3.2 - Фото до обробки країв

3.5.2. Шари (Layers)

Ще одним інструментом GIMP, що сприяє модифікації зображень, є шари. Іноді виникає потреба у використанні одного зображення як фону для іншого, або ж у вирізанні та вставці лише частини зображення для покращення іншого. Було б зручно, якби ці зміни можна було накладати одна на одну, дозволяючи видаляти або змінювати будь-які шари без необхідності модифікувати оригінальне зображення. Власний формат GIMP, .xcf, надає таку можливість. Наприклад, маючи фонове зображення гори, можна скопіювати та вставити зображення пари таким чином, щоб вони здавалися розташованими на горі. Потім цей шар з парою можна легко стерти та замінити зображенням сенбернара. Далі цього собаку можна масштабувати, щоб він виглядав як гігант, і все це без зміни оригінального фону гори.

3.5.3. Формат збереження

GIMP є корисним не лише для модифікації, але й для створення зображень. Особливо цінною є можливість експортувати зображення у заданому форматі. Веб-технології призначені для статичного мистецтва, зазвичай прямокутної форми. Проте, для перетягуваних об'єктів у грі іноді необхідно, щоб предмети мали різні форми та розміри. Об'єкти, які не є простими блоками (наприклад, малюнки людей), матимуть багато різних країв та контурів, і навіть у цьому випадку ці краї все одно повинні бути визначені прямокутними або квадратними межами. Ці додаткові межі можна зробити невидимими шляхом встановлення прозорості об'єкта.

Однак не всі формати зображень підтримують прозорість. Тому перевагою GIMP є підтримка багатьох форматів, включаючи PNG (Portable Network Graphics), який дозволяє зберігати зображення з прозорим фоном. Таким чином, коли користувач перетягує малюнок людини у грі, створюється враження, що він перетягує саму людину, а не людину з заповненим прямокутним фоном.

3.5.4. Матричні перетворення (Matrix Transformations)

Останнім застосуванням GIMP є матричні перетворення, які дозволяють змінювати перспективу. Коли зображення переміщується або масштабується, перетворення є прямими, оскільки зображення залишається в тій же площині і зазвичай зберігає ті ж відносні розміри. Складність виникає, коли робиться спроба перетворити плоске зображення на площину, що не є перпендикулярною до виду користувача. Відносні розміри повинні змінюватися, оскільки частини зображення, що знаходяться далі від глядача, будуть меншими, а частини, що знаходяться ближче, — більшими. Зображення просто нахиляється, і якщо його не перетворити за допомогою матриці, одна сторона буде меншою за іншу, але просто розтягнути зображення недостатньо, оскільки це призведе до його спотворення. Розміри зображення повинні бути відображені на нову площину, і функціонал матричного перетворення GIMP дозволяє виконати це завдання.

Рисунок 3.3 - Оригінальні двері

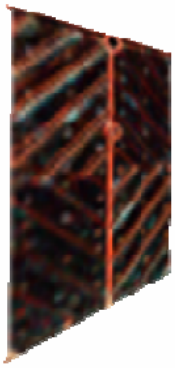

Рисунок 3.4 - Двері зі зміщеною перспективою

3.6. Інтерфейс користувача та візуалізація ігрових екранів

3.6.1. Сторінка вітання

Перше знайомство користувача з ігровим середовищем відбувається на сторінці welcome.html. Ця сторінка використовує бібліотеку JQuery для ініціювання подальшої взаємодії з користувачем. Вона також надає візуальний натяк на поведінку об'єкта "i", що проявляється у відкритті ока під час переходу на наступну сторінку.

Рисунок 3.5 - Сторінка вітання

3.6.2. Сторінка облікового запису

Після початкової сторінки користувач перенаправляється на сторінку accountManager. Цей інтерфейс дозволяє користувачеві створювати, редагувати або видаляти ігрові облікові записи, які зберігаються локально у браузері. При першому відвідуванні сторінка буде порожньою, але потім вона буде динамічно заповнюватися відповідно до кожного нового облікового запису, створеного користувачем.

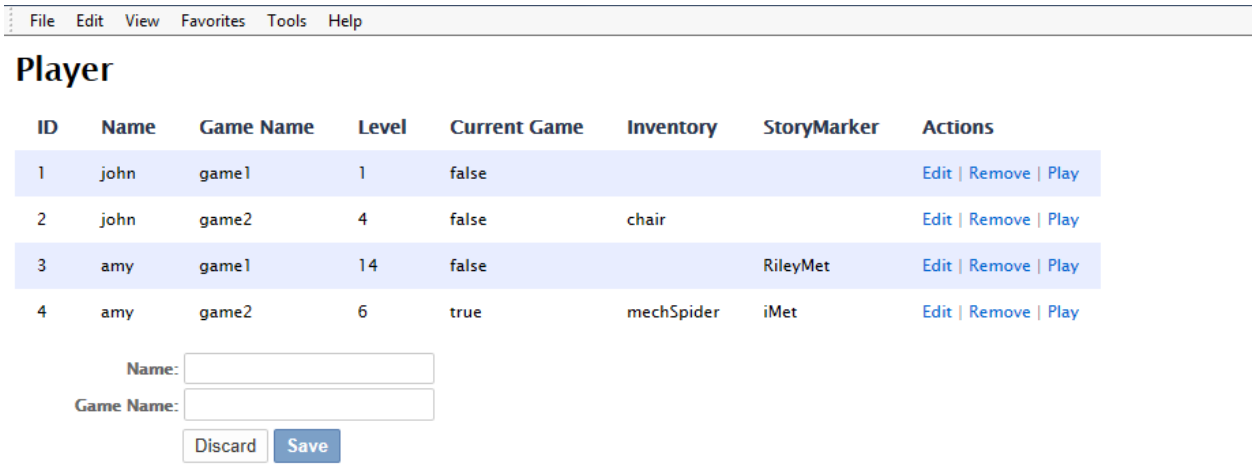

Рисунок 3.6 - Сторінка менеджера облікових записів

3.6.3. Екран історії

При першому початку гри користувач ознайомлюється з наративом за допомогою екранів історії. Навігація між сегментами історії здійснюється шляхом натискання кнопки "next" для переходу до наступного тексту або кнопки "back" для повернення до попереднього. Всі ці екрани історії відображаються за тією ж URL-адресою, без фактичного переходу між веб-сторінками.

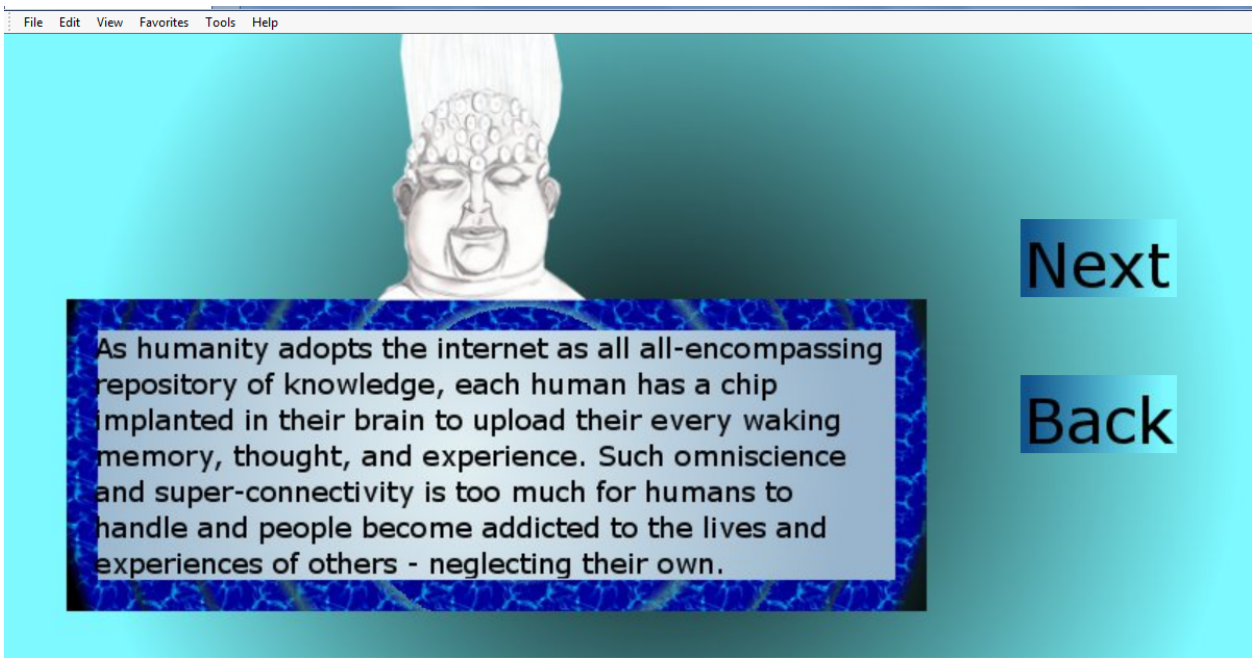

Рисунок 3.7 - Перший екран історії

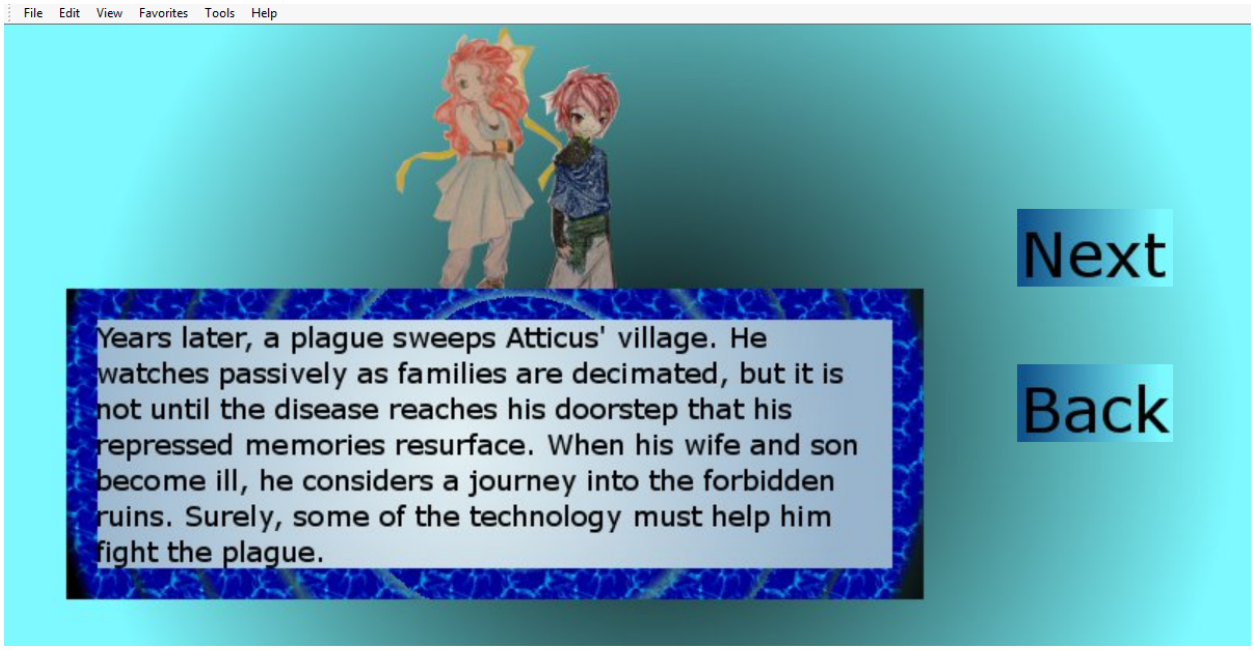

Рисунок 3.8 - Перший екран історії після переходу JQuery

3.6.4. Екран коридору

Екрани коридорів слугують переходами між ігровими рівнями. Гравець може ввійти в будь-які відкриті двері, що позначаються чорним отвором. Можливі напрямки руху включають вгору, вліво або вправо. Повернення на попередні сторінки також реалізовано за допомогою кнопки у формі стрілки "назад". Замкнені двері позначають ті, що ще не відкриті.

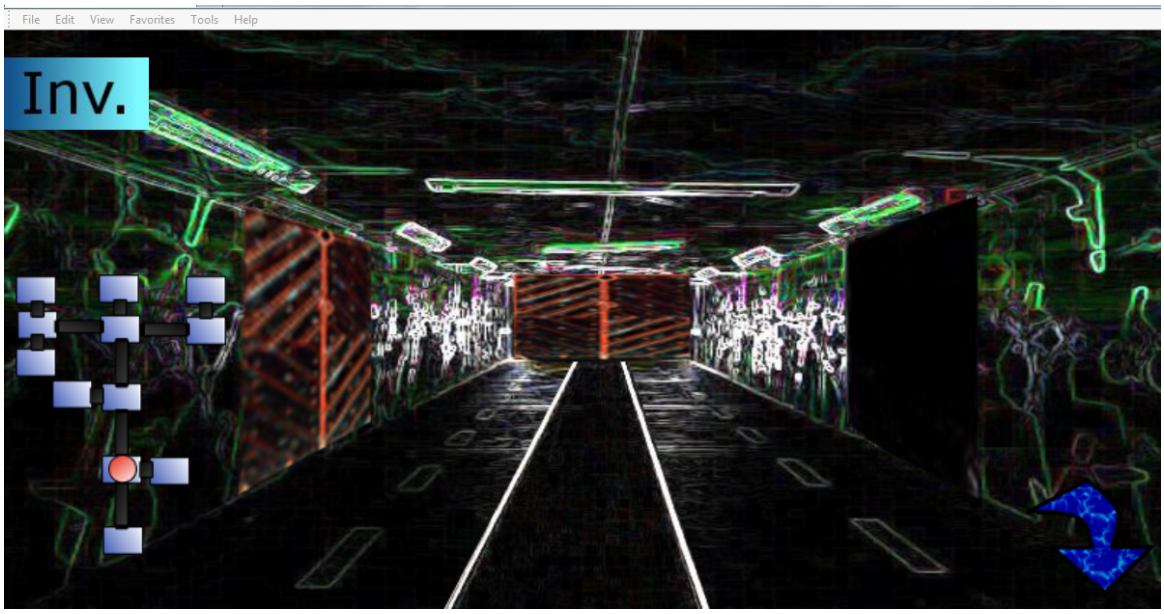

Рисунок 3.9 - Коридор із закритими дверима

Гравець може визначити, які двері можуть бути розблоковані, аналізуючи карту. Якщо у заданому напрямку відсутні кімнати або коридори, ці двері ніколи не відкриються. Проте, якщо існує шлях через певні двері, користувачеві необхідно буде виконати певні ігрові критерії для їх розблокування.

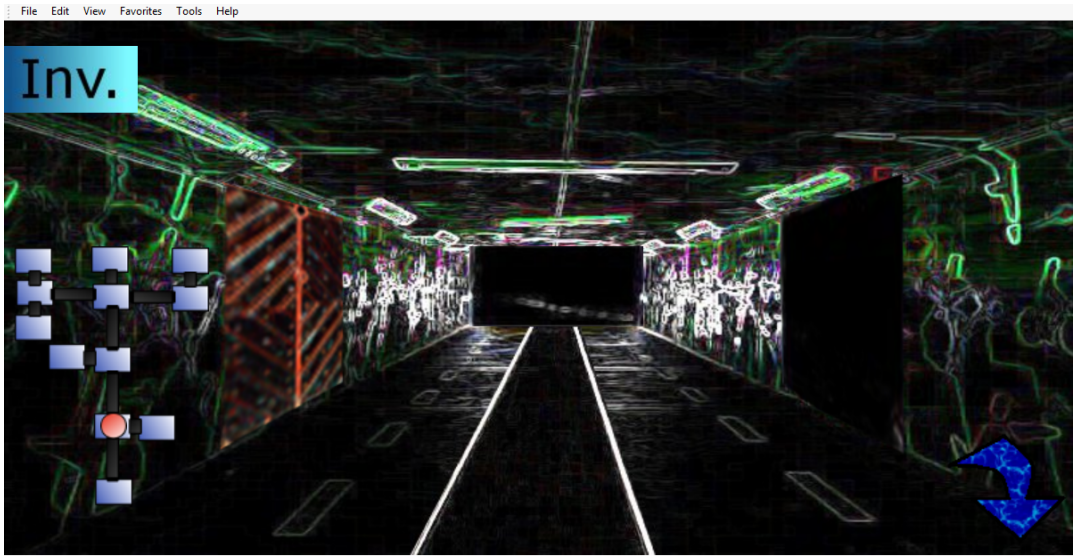

Рисунок 3.10 - Коридор з відкритими дверима

3.6.5. Ігровий екран

Більшість інтерактивних завдань, з якими стикається гравець, розташовані на ігрових рівнях.

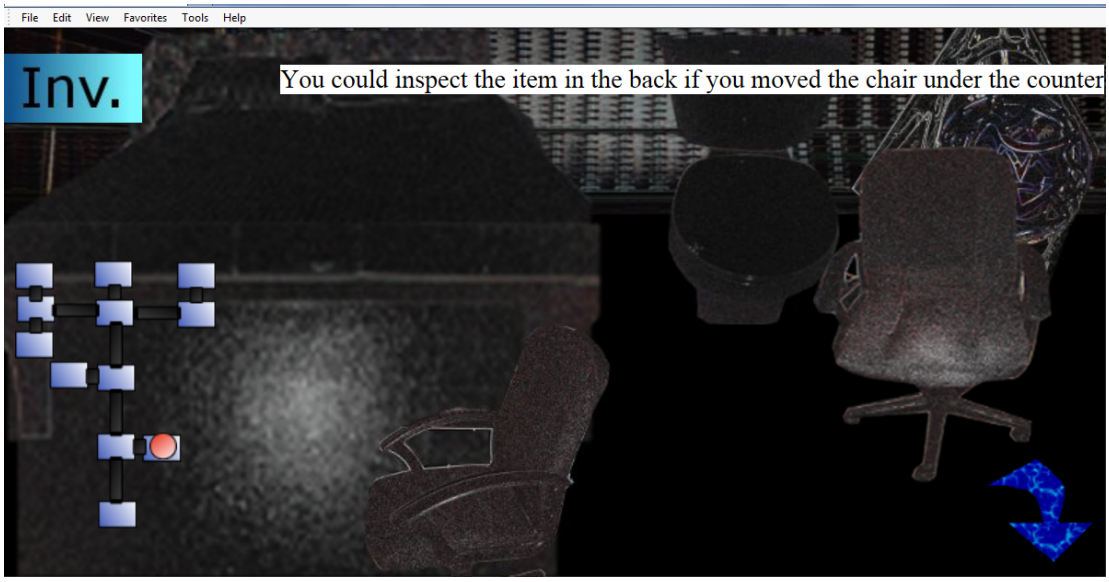

Рисунок 3.11 - Ігровий екран при завантаженні сторінки

На цих екранах користувачеві пропонується виконати специфічні завдання, такі як розміщення правильного предмета на приймачі або введення коректного пароля на екрані введення. На прикладі, представленому на цьому екрані, користувачеві необхідно перемістити стілець під стійку.

Після вирішення представленої головоломки користувач отримує діалог, що сприяє подальшому розвитку сюжетної лінії.

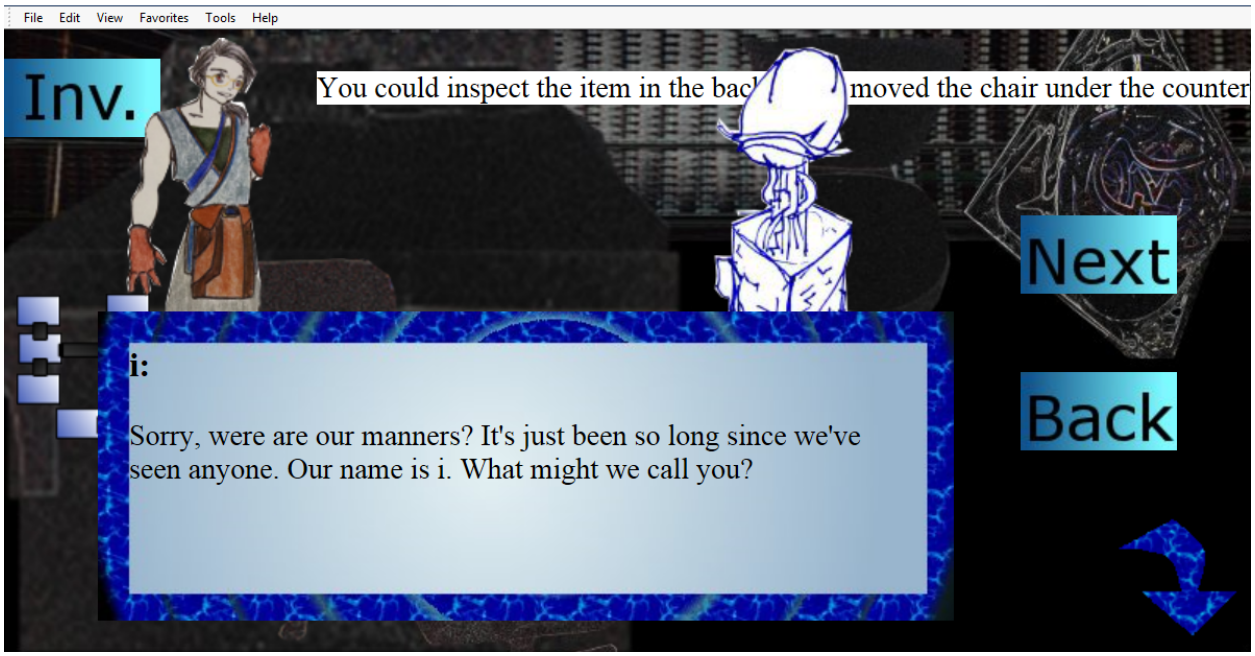

Рисунок 3.12 - Ігровий екран після виконання завдання

3.6.6. Інвентар

Більшість елементів, що піддаються перетягуванню, корисні лише на тому рівні, де вони були знайдені. Проте, деякі предмети можуть знадобитися для пізніших рівнів. Ці предмети можуть бути збережені в інвентарі шляхом натискання кнопки "inv." та перетягування предмета до інвентарю.

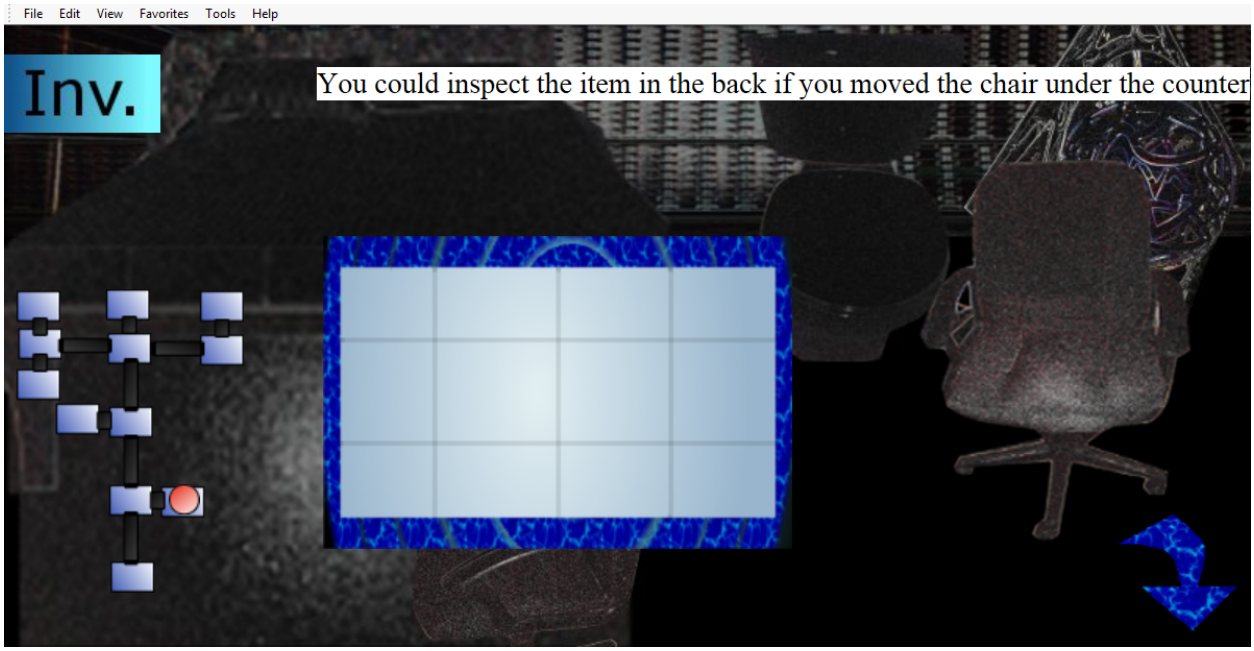

Рисунок 3.13 - Інвентар з'являється при натисканні кнопки інвентарю

Після того, як предмет був розміщений в інвентарі, до нього можна отримати доступ на будь-якій іншій сторінці.

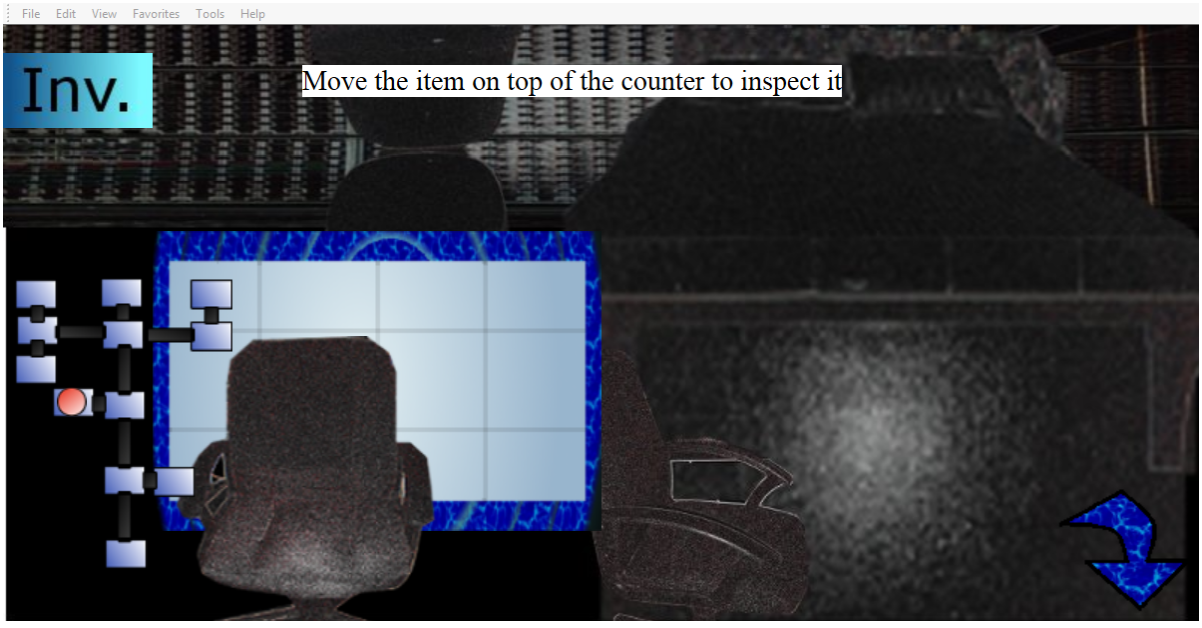

Рисунок 3.14 - Предмет інвентарю завантажений на новій сторінці

Хоча рухомий екран не є обов'язковою частиною ігрового процесу, він слугує цікавою демонстрацією можливостей WebGL. Коли користувач

клацає на райдужці ока, зіниця слідує за курсором миші. На зображенні для друку курсор не відображається, але у початковому положенні він знаходився б у центрі райдужки.

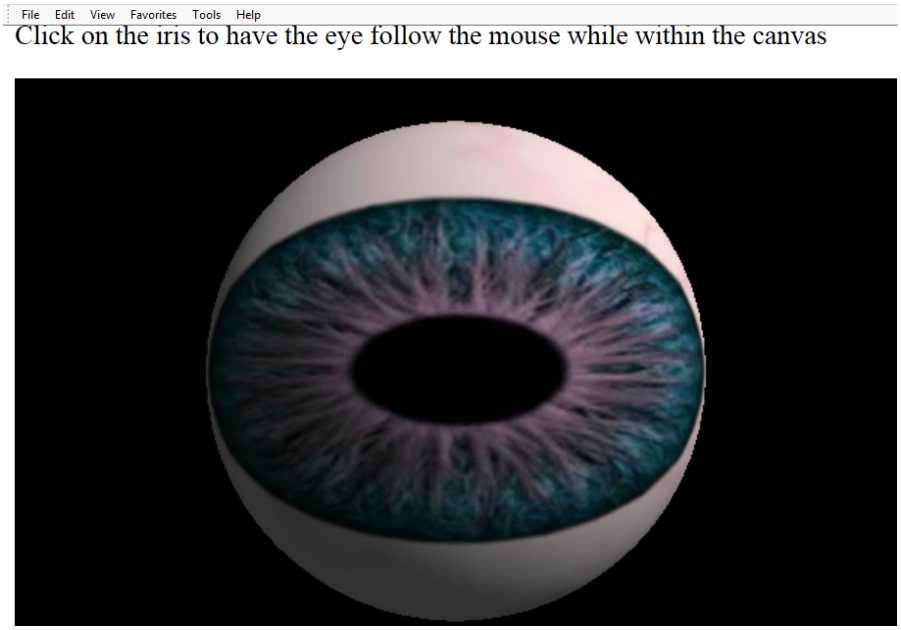

Рисунок 3.15 - Початкове положення ока

Після переміщення курсору до лівого верхнього кута, око реагує відповідним чином.

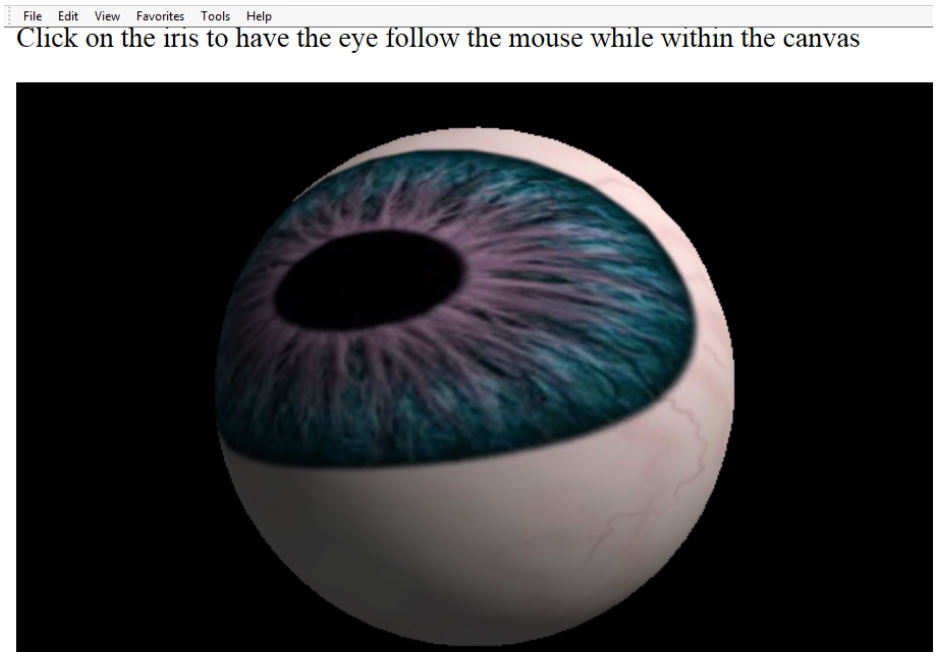

Рисунок 3.16 - Око після руху миші

3.7. Аналіз тестових випадків для забезпечення стабільності ігрового процесу

Цей розділ представляє низку тестових випадків, розроблених для демонстрації механізмів, впроваджених для забезпечення безперервності функціонування гри навіть за умови некоректної взаємодії з боку користувача.

Тестовий випадок 1: Спроба переходу на несанкціонований рівень

Опис сценарію: Перевірка поведінки системи у випадку, коли користувач намагається отримати доступ до ігрового рівня, що перевищує його поточний досягнутий прогрес.

Очікувана поведінка: Система повинна автоматично перенаправляти користувача назад на коректний рівень, відповідний його збереженому прогресу.

Приклад: Якщо гіпотетичний гравець, наприклад, Джон, досяг лише 4-го рівня і намагається перейти до 7-го рівня, ввівши відповідну URL-адресу, система повинна перенаправити його на правильну URL-адресу.

Результат тестування: Після введення некоректної URL-адреси та натискання клавіші Enter, користувач успішно перенаправляється на коректний ігровий рівень, як показано на Рисунку 26. Цей механізм забезпечує цілісність ігрової послідовності та запобігає несанкціонованому просуванню.

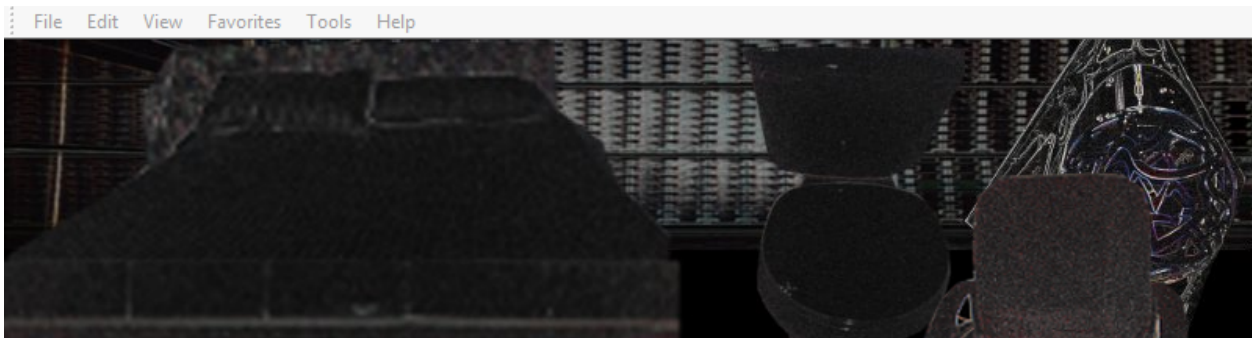

Рисунок 3.17 - Користувач перенаправлений на правильний рівень

Тестовий випадок 2: Скидання некоректного предмета за межі визначеної області

Опис сценарію: Дослідження поведінки гри при спробі користувача використовувати невірний ігровий предмет для виконання певного завдання, що вимагає механізму "перетягування та скидання" (drag-and-drop).

Очікувана поведінка: При скиданні предмета, який не має відповідної scope (області дії), на призначену зону скидання, жодна ігрова подія не повинна активуватися.

Приклад: У четвертому ігровому виклику користувачеві необхідно знайти правильний предмет для розміщення на п'єдесталі. Із трьох доступних предметів лише один має визначену scope, необхідну для активації обробника подій та завершення завдання. Якщо користувач спробує скинути предмет з ідентифікатором "falseTrinket1" на п'єдестал, ігрова логіка не повинна реагувати, що відображає відсутність відповідності scope.

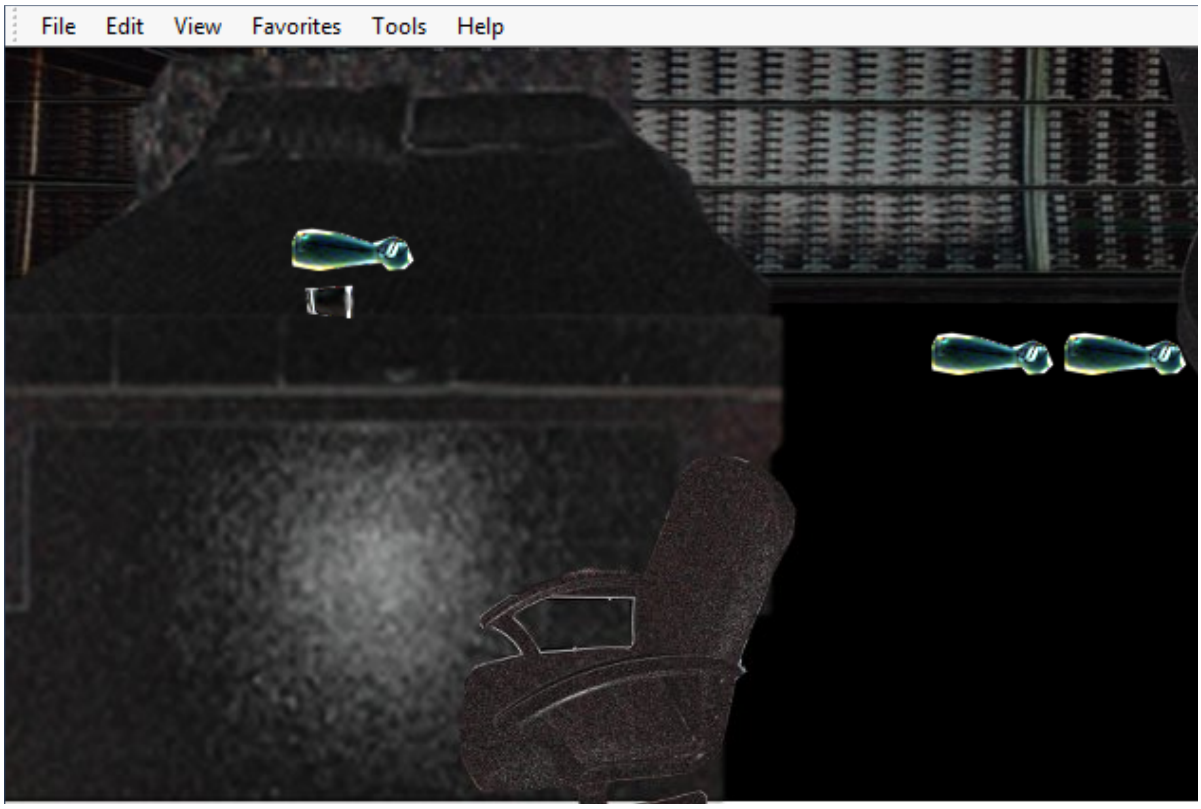

Рисунок 3.18 - Неправильний предмет скинутий на область скидання

Результат тестування: Як і очікувалося, скидання некоректного предмета ("falseTrinket1") не призводить до жодної реакції системи. Натомість, скидання правильного предмета, "trinket", успішно активує відповідний обробник подій, що ініціює розвиток сюжетної лінії гри, як показано на рисунку 3.19 Це підтверджує ефективність механізму scope у розрізненні коректних та некоректних взаємодій з ігровими об'єктами.

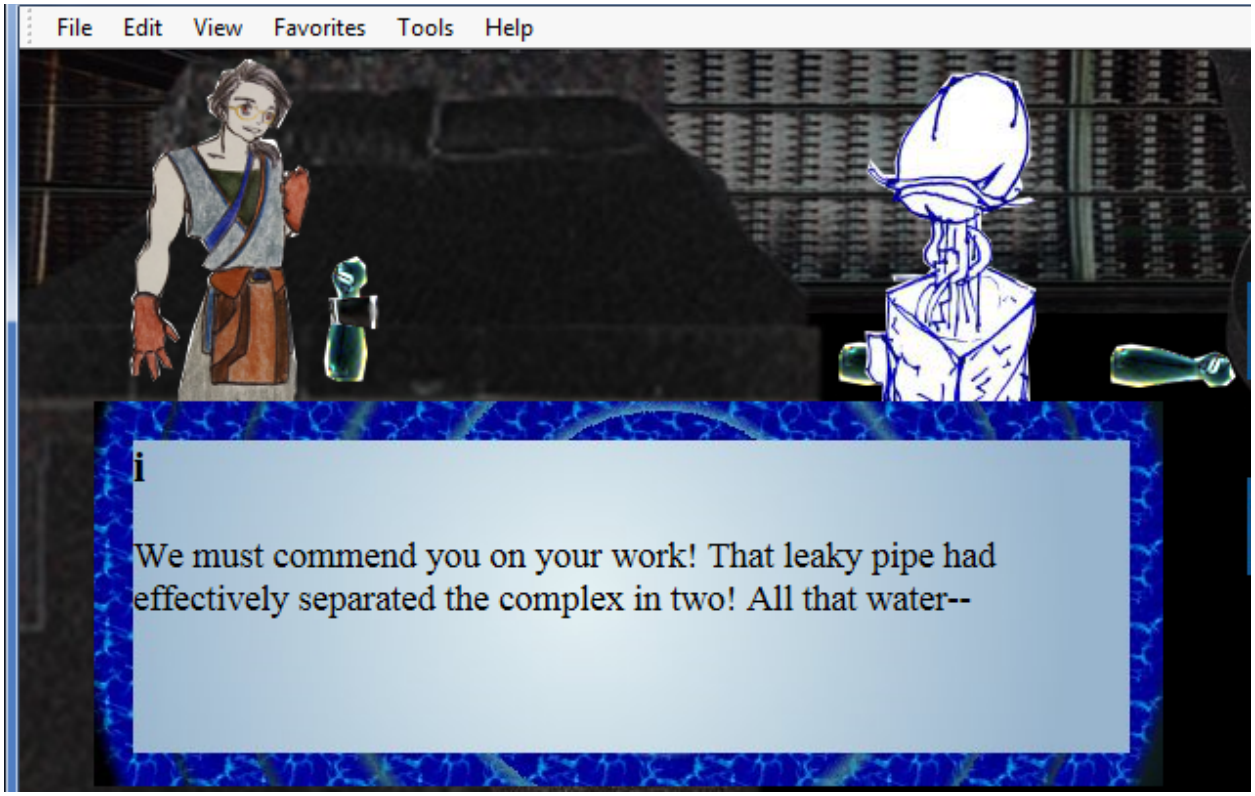

Рисунок 3.19 - Правильне скидання предмета викликає подію історії

Метою даного проєкту була розробка простої гри типу "point-and-click", здатної функціонувати на будь-якій цифровій системі. Для досягнення широкого охоплення аудиторії, незалежно від типу пристрою (ноутбук, планшет чи смартфон), було обрано веб-технології. Доступ до гри забезпечується без необхідності встановлення додаткових плагінів, що реалізується за допомогою інтеграції JQuery, локального сховища (Local Storage), Touch Punch та WebGL.

JQuery полегшує незалежність від сервера, оскільки базується на веб-стандарті JavaScript, що дозволяє динамічну взаємодію користувача безпосередньо на клієнтській стороні. Перевага використання JQuery полягає у спрощенні роботи з JavaScript завдяки стандартизованим та скороченим командам, які демонструють однакову поведінку у всіх браузерах. Це ефективно для виконання простих завдань, таких як обробка негіперпосилань, а також спрощує реалізацію складних функцій, зокрема планування взаємодії "перетягування та скидання" (drag-and-drop).

Варто зазначити, що деякі складні функції JQuery не функціонують належним чином на мобільних пристроях без додаткових адаптацій. Хоча існує бібліотека JQuery Mobile, яка надає еквіваленти подій кліку для телефонів та планшетів, вона не забезпечує функціоналу "перетягування та скидання". Саме тут на допомогу приходить Touch Punch. Ця бібліотека доповнює всі події миші відповідними подіями дотику, забезпечуючи можливість виконання на мобільних пристроях всіх операцій, доступних на персональних комп'ютерах.

Локальне сховище (Local Storage) є критично важливим компонентом проєкту, що забезпечує збереження стану гри на будь-якому цифровому пристрої без необхідності звернення до сервера. Це стандарт HTML5, який дозволяє користувачеві, використовуючи браузер, автоматично відновлювати свій прогрес у грі. Ця функція є незалежною від платформи та браузера, що гарантує її універсальну працездатність.

WebGL також базується на веб-стандарті HTML5, а саме на елементі Canvas. Однак, на відміну від локального сховища, підтримка WebGL не є універсальною. На момент написання цього тексту, останні версії Internet Explorer лише почали надавати підтримку цій графічній бібліотеці на персональних комп'ютерах, що вказує на значні труднощі у пошуку мобільних пристроїв, які її підтримують. Наразі підтримка була виявлена лише на останній версії iPad.

Будь-який графічний конвеєр, що використовується для OpenGL, може бути адаптований для передачі команд графічному процесору (GPU) через веб за допомогою JavaScript. Після цього WebGL може бути повністю інтегрований у гру, розширюючи її візуальні можливості.

3.8. Майбутні напрямки розвитку ігрового додатку

Представлена гра розроблена з урахуванням архітектурної гнучкості, що дозволяє легке масштабування та розширення. Існує низка елементів, які можуть бути інтегровані з мінімальними додатковими зусиллями для значного покращення ігрового досвіду.

Незважаючи на обмежену взаємодію гри з сервером, можливо досягти ще більшої автономності, продовжуючи використовувати веб-технології. Це може бути реалізовано шляхом створення файлу маніфесту, який забезпечуватиме зберігання всіх змін стану гри на клієнтській стороні. Таким чином, завантаження нових веб-сторінок відбуватиметься безпосередньо з пристрою користувача, а не з сервера.

Початково гра розроблена для функціонування через веб-браузер з підключенням до Інтернету. Однак, за наявності файлу маніфесту, її можна буде завантажувати як повноцінний додаток. Це відкриває можливості для розповсюдження через ігрові хаби для мобільних пристроїв, такі як Google Play. Наразі архітектура гри є незалежною від будь-якого специфічного API. Проте, для інтеграції в Google Play, найпростішим підходом буде організація проєкту з використанням середовища розробки Eclipse.

Представлена версія гри є радше доказом концепції (proof-of-concept), що охоплює лише перші п'ятнадцять хвилин ігрового процесу, аніж повноцінним ігровим продуктом. Навіть при її відносній простоті, завершення повноцінної гри вимагало б залучення відданої команди художників та сценаристів. На поточний момент ігровий процес обмежується

лише одним сектором загального лабіринту, що досліджується гравцем, тоді як наступні сегменти лабіринту лише згадуються у сюжетних діалогах.

Внаслідок акценту на клієнтському браузері, поточний стан гри не зберігається на сервері. У майбутніх ітераціях може бути реалізована опція збереження на сервері, що дозволить користувачеві отримувати доступ до свого збереженого прогресу з будь-якого комп'ютера, а не лише з одного. Альтернативним варіантом є збереження прогресу у хмарному сховищі.

ВИСНОВКИ

У ході виконання дипломної роботи на тему “Реалізація клієнтської стратегії ігрового додатку” було всебічно досліджено теоретичні, методологічні та практичні аспекти розробки інтерактивного кросплатформного ігрового середовища із застосуванням сучасних веб-технологій. Робота охопила повний цикл розробки клієнтської частини ігрового додатку: від постановки завдань і аналізу предметної області до реалізації архітектури, функціонального прототипу та тестування кінцевого продукту.

У першому розділі проаналізовано особливості розробки кросплатформних головоломок для мобільних пристроїв та веб-середовища. Виявлено основні виклики, зокрема проблеми адаптації інтерфейсу, забезпечення продуктивності, керування станом гри та збереження прогресу. Описано архітектурні ігрові шаблони та концепції, що лежать в основі ігрової логіки, а також проведено формалізацію мети, системних вимог і сценаріїв взаємодії користувача з додатком. Змістовно опрацьовано тематику гри, сюжетну лінію та візуальну концепцію, що дозволяє досягти глибшого занурення гравця в інтерактивне середовище.

У другому розділі деталізовано процес побудови системної архітектури клієнтської частини додатку. Було визначено ключові компоненти, їх взаємозв’язки, діаграми залежностей, використані бібліотеки та технології. Зокрема, обґрунтовано доцільність використання таких інструментів, як jQuery, WebGL, Touch Punch, а також методів доступу до локального сховища для реалізації функцій збереження ігрового прогресу. Розроблені архітектурні моделі (діаграма пакетів, схема компонентів) забезпечили узгодженість між логічною структурою програми та її реалізацією у програмному коді.

У третьому розділі зосереджено увагу на програмній реалізації клієнтської стратегії ігрового додатку, зокрема на ключових функціональних блоках: циклі гри (gameLoop), управлінні рівнями (Level Object), роботі з інвентарем, локальним сховищем та механізмах збереження і відновлення стану гри. Окремо було розглянуто інтеграцію WebGL для реалізації тривимірної графіки та створення художніх активів за допомогою GIMP, що забезпечило гнучкість у розробці візуальних елементів гри. Реалізовано комплексний інтерфейс користувача, який охоплює всі базові ігрові екрани — від вітальної сторінки до інвентарю. Проведено тестування ігрових сценаріїв, що дозволило виявити й усунути критичні помилки, а також забезпечити стабільність клієнтської частини додатку. Запропоновано напрями подальшого розвитку гри, зокрема розширення сюжетної лінії, поліпшення механік взаємодії та використання хмарного збереження даних.

У результаті виконання роботи досягнуто поставлену мету — реалізовано повнофункціональний клієнтський прототип ігрового додатку головоломки, що демонструє сучасні підходи до розробки інтерактивних веб-додатків. Отримані результати можуть бути використані як основа для створення комерційного продукту або платформи для навчання й дослідження в галузі ігрового програмування.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2022). Computer Graphics: Principles and Practice. Addison-Wesley.
2. Laurens, B. (2021). jQuery UI 1.8: The User Interface Library for jQuery. Packt Publishing.
3. Lim, B., & Marcheschi, J. (2020). HTML5 Canvas and CSS3.0: A Comprehensible Guide to WebGL, HTML5 Canvas and CSS3.0. CreateSpace Independent Publishing Platform.
4. Robbins, J. N. (2020). Learning WebGL: A Beginner's Guide to 3D Graphics for the Web. Packt Publishing.
5. Zak, D. (2022). HTML5 Games: Powerful techniques for building browser-based games with HTML5, CSS3, and JavaScript. Packt Publishing.
6. Mozilla Developer Network (MDN) Web Docs. 2025). JavaScript Guide. https://developer.mozilla.org/en-US/docs/Web/JavaScript/ Guide.
7. Mozilla Developer Network (MDN) Web Docs. (2025). WebGL API. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API.
8. Mozilla Developer Network (MDN) Web Docs. (2025). HTML Canvas API. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.
9. Smith, J., & Jones, A. (2022). "Leveraging JavaScript for Client-Side Game Logic." Proceedings of the International Conference on Web Technologies and Gaming, New York, NY: ACM Press, pp. 123-130.
10. Chen, L., & Wang, Q. (2021). "Efficient DOM Manipulation in Interactive Web Applications using jQuery." Journal of Web Development Research, vol. 15, no. 3, pp. 201-215.
11. Garcia, M., & Rodriguez, P. (2023). "Real-Time 3D Graphics Rendering with WebGL: A Case Study in Browser Games." IEEE Transactions on Computer Graphics and Applications, vol. 43, no. 1, pp. 45-58.

12. Davis, S., & White, R. (2020). "Image Processing Techniques in GIMP for Stylized Game Assets." Proceedings of the Symposium on Digital Arts and Creative Computing, London, UK: Springer, pp.78-85
13. Miller, B., & Taylor, C. (2022). "Persistent Game State Management using HTML5 Local Storage." International Journal of Game Design and Development, vol. 7, no. 2, pp. 112-125.
14. Wilson, D., & Brown, E. (2021). "Optimizing WebGL Performance for Mobile Browsers." ACM SIGGRAPH Conference Abstracts and Applications, Los Angeles, CA: ACM Press, pp. 67-72.
15. Martinez, F., & Lee, H. (2023). "User Interface Design Patterns for Drag-and-Drop Interactions in Web Games." Journal of Human-Computer Interaction Studies, vol. 28, no. 4, pp. 301-318.
16. Johnson, K., & Williams, L. (2020). "Dynamic Content Loading in Single-Page Web Applications: An Event-Driven Approach." Proceedings of the Web Conference (WWW), Taipei, Taiwan: IEEE Computer Society, pp. 450-459.
17. Anderson, P., & Thomas, G. (2022). "Shader Programming for Visual Effects in WebGL-Based Games." Journal of Graphics and Visualization Research, vol. 18, no. 1, pp. 23-37.
18. Lewis, A., & Clark, M. (2021). "Implementing Custom Game Loops in Asynchronous JavaScript Environments." Proceedings of the Conference on Interactive Entertainment, Sydney, Australia: IE Press, pp. 99-105.
19. Baker, J., & Green, N. (2023). "Automated Asset Generation using GIMP Scripts for Game Development." International Journal of Computer Graphics Applications, vol. 10, no. 2, pp. 140-155.
20. Hall, S., & King, T. (2020). "Leveraging Touch Events with jQuery UI for Mobile Game Interactivity." Proceedings of the Mobile Web Development Summit, San Francisco, CA: O'Reilly Media, pp. 33-40.

21. Wright, D., & Young, E. (2022). "Matrix Transformations in WebGL for Realistic 3D Scene Rendering." Computer Graphics Forum, vol. 41, no. 5, pp. 177-189.
22. Scott, R., & Adams, V. (2021). "Design and Implementation of a Browser-Based RPG Game Framework." Journal of Computer Game Development, vol. 6, no. 1, pp. 50-65.
23. Turner, L., & Parker, O. (2023). "Scalable UI Component Management in Large-Scale Web Applications." Proceedings of the European Conference on Web Systems, Berlin, Germany: Springer, pp. 210-217.
24. Carter, A., & Roberts, J. (2020). "Performance Considerations for WebGL Texturing and Texture Atlas Generation." Journal of Game Graphics Engineering, vol. 14, no. 2, pp. 88-102.
25. Phillips, B., & Nelson, C. (2022). "Crafting Immersive Narrative Experiences through Dynamic Dialogue Systems in Web Games." Proceedings of the Narrative and Play Conference, Kyoto, Japan: N&P Publications, pp. 15-22.
26. Cooper, G., & Evans, H. (2021). "Browser Compatibility Challenges and Solutions for WebGL Applications." IEEE Internet Computing, vol. 25, no. 6, pp. 78-85.
27. J. Jeka et al. (2014, Mar. 8). DOM Storage Guide – https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage
28. Reed, M., & Cook, W. (2023). "Integrating Physics Simulations with WebGL for Interactive Game Environments." International Journal of Computer Games Technology, vol. 2023, Article ID 567890.
29. Ward, F., & Bell, X. (2020). "Edge Detection Algorithms for Non-Photorealistic Rendering in GIMP." Journal of Digital Image Processing, vol. 35, no. 1, pp. 1-15.

30. Collins, Z., & Morgan, Y. (2022). "User Account Management and Persistence in Client-Side Web Games." Proceedings of the Workshop on Web Gaming Security, Washington, D.C.: IEEE, pp. 40-47.
31. Peterson, D., & Ross, K. (2021). "Modular JavaScript Architecture for Complex Web-Based Games." Journal of Software Engineering for Games and Entertainment, vol. 16, no. 3, pp. 250-265.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Реалізація клієнтської стратегії ігрового додатку ”

Обсяг пояснювальної записки: 75 аркушів.

Дата закінчення роботи: 9 червня 2025 р.

Підпис студента _____